# HP Operations エージェント

Windows®、HP-UX、Linux、Solaris、AIX向け

ソフトウェアバージョン: 11.10

## ユーザーガイド

ドキュメントリリース日 : 2012年8月

ソフトウェアリリース日 : 2012年8月

# ご注意

## 保証

HP製品、またはサービスの保証は、当該製品、およびサービスに付随する明示的な保証文によってのみ規定されるものとします。ここでの記載で追加保証を意図するものは一切ありません。ここに含まれる技術的、編集上の誤り、または欠如について、HPはいかなる責任も負いません。

ここに記載する情報は、予告なしに変更されることがあります。

## 権利の制限

機密性のあるコンピュータソフトウェアです。これらを所有、使用、または複製するには、HPからの有効な使用許諾が必要です。商用コンピュータソフトウェア、コンピュータソフトウェアに関する文書類、および商用アイテムの技術データは、FAR12.211および12.212の規定に従い、ベンダーの標準商用ライセンスに基づいて米国政府に使用許諾が付与されます。

## 著作権について

© Copyright 2010 - 2012 Hewlett-Packard Development Company, L.P.

## 商標について

Adobe™は、Adobe Systems Incorporated (アドビシステムズ社) の商標です。

Microsoft®およびWindows®は、米国におけるMicrosoft Corporationの登録商標です。

UNIX®は、The Open Groupの登録商標です。

本製品には 'zlib' 汎用圧縮ライブラリのインタフェースが使用されています。'zlib': Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

# ドキュメントの更新情報

このマニュアルの表紙には、以下の識別情報が記載されています。

- ソフトウェアのバージョン番号は、ソフトウェアのバージョンを示します。
- ドキュメントリリース日は、ドキュメントが更新されるたびに変更されます。
- ソフトウェアリリース日は、このバージョンのソフトウェアのリリース期日を表します。

最新の更新のチェック、またはご使用のドキュメントが最新版かどうかのご確認には、次のサイトをご利用ください。

**http://support.openview.hp.com/selfsolve/manuals**

このサイトを利用するには、HP Passportへの登録とサインインが必要です。HP Passport IDの取得登録は、次のWebサイトから行なうことができます。

**http://h20229.www2.hp.com/passport-registration.html(英語サイト)**

または、HP Passport のログインページの [**New users - please register**] リンクをクリックします。

適切な製品サポートサービスをお申し込みいただいたお客様は、更新版または最新版をご入手いただけます。詳細は、HPの営業担当にお問い合わせください。

# サポート

HPソフトウェアサポートオンラインWebサイトを参照してください。

**http://support.openview.hp.com**

HPソフトウェアが提供する製品、サービス、サポートに関する詳細情報をご覧いただけます。

HPソフトウェアオンラインではセルフソルブ機能を提供しています。お客様の業務の管理に必要な対話型の技術支援ツールに素早く効率的にアクセスいただけます。HPソフトウェアサポートWebサイトのサポート範囲は次のとおりです。

- 関心のある技術情報の検索
- サポートケースとエンハンスメント要求の登録とトラッキング
- ソフトウェアパッチのダウンロード
- サポート契約の管理
- HP サポート窓口の検索
- 利用可能なサービスに関する情報の閲覧
- 他のソフトウェアカスタマーとの意見交換
- ソフトウェアトレーニングの検索と登録

一部を除き、サポートのご利用には、HP Passportユーザとしてご登録の上、ログインしていただく必要があります。また、多くのサポートのご利用には、サポート契約が必要です。HP Passport IDの登録は、次の場所で行います。

**http://h20229.www2.hp.com/passport-registration.html(英語サイト)**

アクセスレベルに関する詳細は、以下のWebサイトにアクセスしてください。

**http://support.openview.hp.com/access_level.jsp**

# 目 次


ユーザー ガイド 1

目 次 5

はじめに 22

ドキュメント マップ 22

データ収集の管理 24

ログ ファイルベースのデータ ストアの使用 24

ログ ファイルの収集 24

logglob 26

logappl 26

logproc 26

logpcmd 26

logdev 27

logtran 27

logls 27

logindx 27

scope のステータス 27

parm ファイル 27

parm ファイルの変更 28

parm ファイルのパラメータ 29

パラメータの説明 33

ID 34

log 34

Thresholds 35

procthreshold 36

subprocinterval 37

gapapp 37

fstypes 38

wait 39

size 39
mainttime 39
days 40
javaarg 40
Flush 41
zone_app 41
project_app 41
proclist 41
appproc 42
proccmd 42
ignore_mt 42
cachemem 42
アプリケーション定義パラメータ 43
データ記録間隔の設定 43
vMA ノードでのデータ収集の設定 43
ハイパースレッディング/マルチスレッディングが有効なシステムにおける CPU メトリックの正規化 44
メトリックのリスト 44
コアに基づいた正規化によって計算されたメトリックの記録 47
データ収集の終了と再開始 48
データ収集の終了 48
データ収集の再開始 49
夏時間 49
システムの時刻の手動変更 50
効果的なデータ収集管理 50
ログファイルにより使用されるディスクスペースの制御 50
mainttime の設定 50
ログファイルの最大サイズの設定 51
サイズ変更プロセスの管理 51
データのアーカイブ 52
アーカイブ処理の管理 52
HP Operations エージェント の操作 54

監視エージェントの設定 54
MIB オブジェクトを監視するエージェントの構成 54
監視対象オブジェクトの永続化 55
監視対象の値を保存しないようにエージェントを設定するには 55
イベント インターセプタの構成 56
RTMA コンポーネントの構成 58
デフォルト設定を変更するには 59
アクセスの制限 60
エージェント ユーザーの設定 61
デフォルトではないユーザーを使用するための要件 61
インストール中のエージェント ユーザーの設定 62
インストール後のエージェント ユーザーの設定 65
Windows でのデフォルト ユーザーの変更 65
プロファイル ファイルの使用 65
もう一方の方法: ovswitchuser コマンドの使用 66
UNIX/Linux でのデフォルト ユーザーの変更 69
プロファイル ファイルの使用 69
もう一方の方法: ovswitchuser コマンドの使用 70
コマンドのデフォルト ユーザーの変更 72
セキュリティコンポーネント変数の設定 72
vMA ノードの監視用 viserver の設定 73
viserver.properties 74
port 74
hosts 74
instance 74
jvmArgs 74
log4jInterval 75
VILog4j.xml 75
Windows でのアプリケーションおよびサービス ログの監視 75
HPOM for Windows からのアプリケーション ログおよびサービス イベント ログの監視 76
UNIX/Linux 9.1x 上の HPOM からのアプリケーション ログおよびサービス イベント ログの監視 77

HPOM for UNIX 8.35 からのアプリケーション ログおよびサービス イベント ログの監視 78
リモート ノードの監視 78
utility プログラムの使い方 80
utility プログラムの実行 80
対話型モードの使い方 81
対話型モードとバッチ モードの使用例 81
utility のコマンド ライン インターフェイス 82
コマンド ライン インターフェイスの使用例 83
utility 走査レポートの詳細 84
走査レポート情報 85
初期値 85
初期 parm ファイルのグローバル情報 85
初期 parm ファイルのアプリケーション定義 86
年代順の詳細 86
parm ファイルのグローバル変更通知 86
parm ファイルのアプリケーション追加通知および削除通知 87
scope のオフタイム通知 87
アプリケーション固有の要約レポート 87
要約 88
プロセスの記録理由の要約 88
走査の開始および終了 89
アプリケーションの全体的な要約 89
コレクタの適用範囲の要約 90
ログ ファイルの内容の要約 90
ログ ファイルの空きスペースの要約 91
utility のコマンド 94
analyze 95
checkdef 96
detail 97
exit 97
guide 97

help 98
list 98
logfile 99
menu 101
parmfile 102
quit 102
resize 102
resize コマンド のレポート 105
例 106
scan 106
sh 108
show 108
start 109
stop 111
extract プログラムの使い方 114
extract プログラムの実 行 115
構 文 115
対 話 型 モード の使 い方 115
extract のコマンド ライン インターフェイス 116
エクスポート関 数 の概 要 119
データのエクスポート方 法 119
エクスポート タスクのサンプル 121
印 刷 可 能 な CPU レポート の生 成 121
カスタマイズしたエクスポート ファイルの作 成 121
データ ファイルのエクスポート 121
エクスポート テンプレート ファイルの構 文 122
パラメータ 124
エクスポート ファイルのタイト ル 126
カスタム グラフまたはレポート の作 成 126
エクスポート ファイルの出 力 127
ASCII フォーマットとdatafile フォーマットに関 する注 意 128
binary フォーマットに関 する注 意 129

バイナリ ヘッダー レコード レイアウト 129
Binary Title Record 129
Binary Item Identification Record 130
Binary Scale Factor Record 130
特 殊 な 倍 率 130
Application Name Record 131
Transaction Name Record 131
Disk Device Name Record 131
Logical Volume Name Record 131
Netif Name Record 132
extract のコマンド 134
application 137
class 138
configuration 139
cpu 140
disk 141
exit 142
export 142
extract 144
filesystem 146
global 147
guide 148
help 149
list 149
logfile 150
lvolume 152
menu 153
monthly 155
netif 156
output 157
process 159
quit 160

report 160
sh 161
shift 161
show 162
start 164
stop 165
transaction 167
weekdays 168
weekly 169
yearly 170
cpsh プログラムの使 用 174
インタラクティブ モード の使 用 174
リアルタイム メトリックの表 示 175
メトリック クラスの変 更 175
すべての使 用 可 能 メトリックの表 示 175
メトリック クラスの整 理 176
メトリックのヘルプの表 示 176
要 約 メトリック データの表 示 176
パフォーマンス アラーム 178
アラームの処 理 178
アラーム ジェネレータ 178
Network Node Manager への SNMP トラップの送 信 179
HPOM へのメッセージの送 信 179
ローカル アクションの実 行 179
アラーム処 理 のエラー 180
アラームの履 歴 データの分 析 181
履 歴 データのアラーム情 報 の例 181
アラーム定 義 の構 成 要 素 182
アラーム構 文 のリファレンス 182
アラーム構 文 182
表 記 183
共 通 要 素 183

コメント 184
複 合 文 184
条 件 184
定 数 185
式 185
メトリック名 185
メッセージ 186
ALARM 文 187
構 文 187
使い方 188
例 189
ALERT 文 190
構 文 190
使い方 191
例 191
EXEC 文 191
構 文 191
使い方 192
例 192
PRINT 文 193
構 文 193
例 193
IF 文 193
構 文 193
使い方 194
例 194
LOOP 文 194
構 文 195
使い方 195
例 195
INCLUDE 文 196
構 文 196

使い方 196
例 196
USE 文 196
構文 196
使い方 197
VAR 文 198
構文 198
使い方 198
例 198
ALIAS 文 199
構文 199
使い方 199
例 199
SYMPTOM 文 199
構文 200
使い方 200
例 200
アラーム定義の例 201
CPU の問題の例 201
スワップ利用率の例 201
時間に起因するアラームの例 201
ディスク インスタンスのアラームの例 202
アラーム定義のカスタマイズ 202
RTMA コンポーネントのアドバイザ 204
アラームと兆候 204
adviser スクリプトでの作業 204
アドバイザの使用 205
複数のシステム上での adviser スクリプトの実行 205
アドバイザ構文 206
構文表記 206
コメント 206
条件 206

定 数 207
式 207
アド バイザ構 文 内 のメトリック名 207
出 力リスト 209
変 数 209
アド バイザ構 文 209
ALARM 文 209
ALERT 文 210
ALIAS 文 210
ASSIGNMENT 文 211
COMPOUND 文 211
EXEC 文 211
IF 文 212
LOOP 文 212
PRINT 文 213
SYMPTOM 文 213
Windows での Performance Collection Component の使 用 216
データ型 とクラス 217
要 約レベル 217
抽 出 またはエクスポートするデータの範 囲 217
ログ ファイル データの抽 出 218
ログ ファイル データのエクスポート 219
ファイル属 性 220
ファイル フォーマット 220
欠 落 値 220
フィールド 区 切り記 号 220
分 単 位 の要 約 インターバル 221
ヘッダー 221
複 数 レイアウト 221
エクスポート ファイルのタイト ル 221
エクスポート ファイルのテンプレート 222
デフォルト のエクスポート ファイル 222

scopent データ 222
DSI データ 223
クイック エクスポート テンプレートの作成 224
エクスポートするメトリックの選択 225
選択内容の保存 225
エクスポート テンプレートの構成 225
エクスポートするメトリックの選択 226
選択内容の保存 227
ログ ファイル データのアーカイブ 227
アーカイブ期間 228
アーカイブ データの追加 228
アーカイブに関するヒント 228
ログ ファイルの分析 229
分析するデータの範囲 230
分析レポート 230
ログ ファイルの走査 230
ログ ファイルのサイズ変更 231
ユーザー オプションの構成 233
収集パラメータの設定 235
アラーム定義の設定 236
データ ソースの構成 237
データ ソース ファイルのフォーマット 237
リモート ロケーションからのデータ ソースの構成 238
トランザクションの構成 238
Persistent DSI Collections の構成 240
Performance Collection Component のステータス確認 241
実行中のプロセス 241
データコム サービス 242
システム サービス 242
システム構成 242
ファイル バージョン番号 242
ステータス ファイルの最新エントリ 242

ステータス ファイルの警 告 とエラー 242
パフォーマンス カウンタ収 集 の構 築 242
パフォーマンス カウンタ収 集 の構 築 243
パフォーマンス カウンタ収 集 の管 理 243
拡 張 収 集ビルダ/マネージャの使 用 のヒント 243
コマンド ラインからの ECBM の管 理 244
データ ソース統 合 の概 要 246
DSI の動 作 246
クラス仕 様 の作 成 247
データの収 集 と記 録 247
データの使 用 247
データ ソース統 合 の使 用 方 法 248
データ収 集 の計 画 248
ログ ファイル フォーマット の定 義 248
ログ ファイルの編 成 方 法 249
ログ ファイル セット の作 成 250
クラスの仕 様 ファイルとログ処 理 のテスト (オプション) 250
ログ ファイル セット へのデータの記 録 251
ログ データの使 用 251
DSI クラスの仕 様 254
クラスの仕 様 254
クラスの仕 様 構 文 254
クラスの記 述 255
構 文 255
デフォルト の設 定 256
CLASS 256
LABEL 256
INDEX BY、MAX INDEXES、ROLL BY 257
ログ ファイル サイズの制 御 260
RECORDS PER HOUR 261
CAPACITY 262
メトリックの記 述 263

METRICS 263
LABEL 264
要 約 方 法 265
PRECISION 266
TYPE TEXT LENGTH 266
クラス仕 様 のサンプル 267
DSI プログラムに関 する参 考 資 料 270
sdlcomp コンパイラ 270
コンパイラ構 文 270
コンパイラ出 力 のサンプル 271
構 成 ファイル 274
DSI メトリックのアラーム定 義 274
アラーム処 理 275
dsilog によるログ処 理 275
構 文 275
dsilog によるデータ処 理 方 法 277
sdlgendata によるログ処 理 のテスト 277
フォーマット ファイルの作 成 280
クラス仕 様 の変 更 281
DSI データのエクスポート 281
DSI ログファイル データをエクスポートする extract プログラムの使 用 例 282
Performance Manager でのデータの表 示 282
sdlutil によるデータの管 理 282
構 文 283
データ ソースの統 合 例 284
dsilog スクリプトの記 述 284
例 1 - 問 題 となる dsilog スクリプト 284
例 2 - 推 奨 する dsilog スクリプト 284
vmstat データの記 録 285
クラス仕 様 のファイルの作 成 285
クラス仕 様 ファイルのコンパイル 286
dsilog によるログ処 理 の開 始 287

データへのアクセス 287
単一ファイルの sar データの記録 288
クラス仕様のファイルの作成 288
クラス仕様ファイルのコンパイル 291
DSI ログ処理の開始 292
複数ファイルの sar データの記録 293
クラス仕様のファイルの作成 294
複数のクラスの仕様ファイルのコンパイル 301
DSI ログ処理の開始 301
複数オプションの sar データの記録 302
システム ユーザー数の記録 312
エラー メッセージ 314
SDL エラー メッセージ 314
DSILOG エラー メッセージ 314
一般エラー メッセージ 315
トランザクション追跡の紹介 318
パフォーマンス管理の改善 318
トランザクション追跡の利点 318
ユーザーの視点から見たトランザクション時間 318
トランザクション データ 319
サービス レベルの目標 319
リアルタイムでの注文処理のシナリオ 319
リアルタイム注文処理の必要事項 320
注文処理アプリケーションの準備 320
トランザクション データの監視 320
ARM の使用のガイドライン 321
トランザクション追跡の動作 324
ARM 2.0 のサポート 324
ARM API コールのサポート 324
arm_complete_transaction コール 325
ARM 組み込みアプリケーションの例 325
アプリケーション名とトランザクション名の指定 326

トランザクション追 跡 デーモン (ttd) 326
ARM API コール ステータス リターン 327
測 定 インターフェイス デーモン (midaemon) 328
トランザクション構 成 ファイル (ttd.conf) 328
新しいアプリケーションの追 加 328
新しいトランザクションの追 加 329
range 値 または slo 値 の変 更 329
構 成 ファイルのキーワード 329
tran 329
range 330
slo 330
構 成 ファイルのフォーマット 331
構 成 ファイル例 332
ARM を使 用 する際 のオーバーヘッド の考 慮 事 項 333
ガイド ライン 333
ディスク I/O オーバーヘッド 334
CPU オーバーヘッド 334
メモリー オーバーヘッド 334
トランザクションについて 336
開 始 する前 に 336
トランザクション追 跡 のセット アップ 336
サービス レベルの目 標 (SLO) の定 義 337
parm ファイルの変 更 337
トランザクション データの収 集 338
エラーの対 応 338
固 有 のトランザクションの制 限 338
構 成 ファイルのカスタマイズ (オプション) 339
パフォーマンス データの監 視 340
アラーム 341
トランザクション追 跡 のメッセージ 341
トランザクション メトリック 342
トランザクション追 跡 の例 344

リアルタイム注文処理の擬似コード 344
構成ファイル例 347
例 1 (注文処理の擬似コード例) 347
例 2 347
例 3 347
例 4 348
高度な機能 350
データ型の使用方法 350
ユーザー定義メトリック 351
scope の組み込み 352
トランザクション ライブラリ 354
ARM ライブラリ (libarm) 354
プラットフォームごとの C コンパイラ オプションの例 358
ARM NOP ライブラリ 359
Java ラッパーの使用 359
例 360
アプリケーションの設定 (arm_init) 360
構文: 360
トランザクションの設定 (arm_getid) 360
UDM を使用したトランザクションの設定 360
メトリックの追加 360
メトリック データの設定 361
UDM を使用しないトランザクションの設定 361
トランザクション インスタンスの設定 362
トランザクション インスタンスの開始 (arm_start) 362
相関係数を使用したトランザクション インスタンスの起動 362
相関係数の要求 362
親相関係数の送信 362
親相関係数の要求と送信 363
相関係数情報の取得 363
相関係数を使用しないトランザクション インスタンスの起動 363
トランザクション インスタンス データの更新 363

UDM を使 用したトランザクション インスタンス データの更 新 363
UDM を使 用しないトランザクション インスタンス データの更 新 364
大 規 模 で非 透 過 的 なアプリケーション プライベート バッファの提 供 364
トランザクション インスタンスの停 止 (arm_stop) 364
メトリックを更 新 するトランザクション インスタンスの停 止 364
メトリックを更 新 しないトランザクション インスタンスの停 止 365
完 全 なトランザクションの使 用 365
UDM を使 用した完 全 なトランザクションの使 用 365
UDM 使 用しない完 全 なトランザクションの使 用 365
その他 のド キュメント 365
記 録 とトレース 366
記 録 366
記 録 ポリシーの設 定 367
トレース 367
アプリケーションの特 定 367
トレース タイプの設 定 369
トレース設 定 ファイルの概 要 370
構 文 370
設 定 ファイルの作 成 370
コマンド ライン ツールを使 用したトレースの有 効 化 とトレース メッセージの表 示 370
トレース GUI を使 用したトレースの有 効 化 とトレース メッセージの表 示 371
トレース機 構 の有 効 化 371
トレース メッセージの表 示 372
トレース リスト ビューの使 用 374
手 順 のツリー ビューの使 用 375
トレースのフィルタリング 375
トラブルシューティング操 作 378
運 用 監 視コンポーネント 378
Performance Collection Component 380
RTMA 381

# はじめに

HP Operations エージェントは、多数のサービス、プロセス、ユーティリティをシステムに導入します。エージェントの動作を設定したり、そのパフォーマンスを監視するときは、コマンド ライン ユーティリティが役立ちます。特定のコマンド ライン ユーティリティを利用することで、エージェントが取り込んだリアルタイムのシステム パフォーマンス データを確認できます。トレース ツールのようなユーティリティは、トラブルシューティング時にエージェントの診断情報を確認する際に役立ちます。

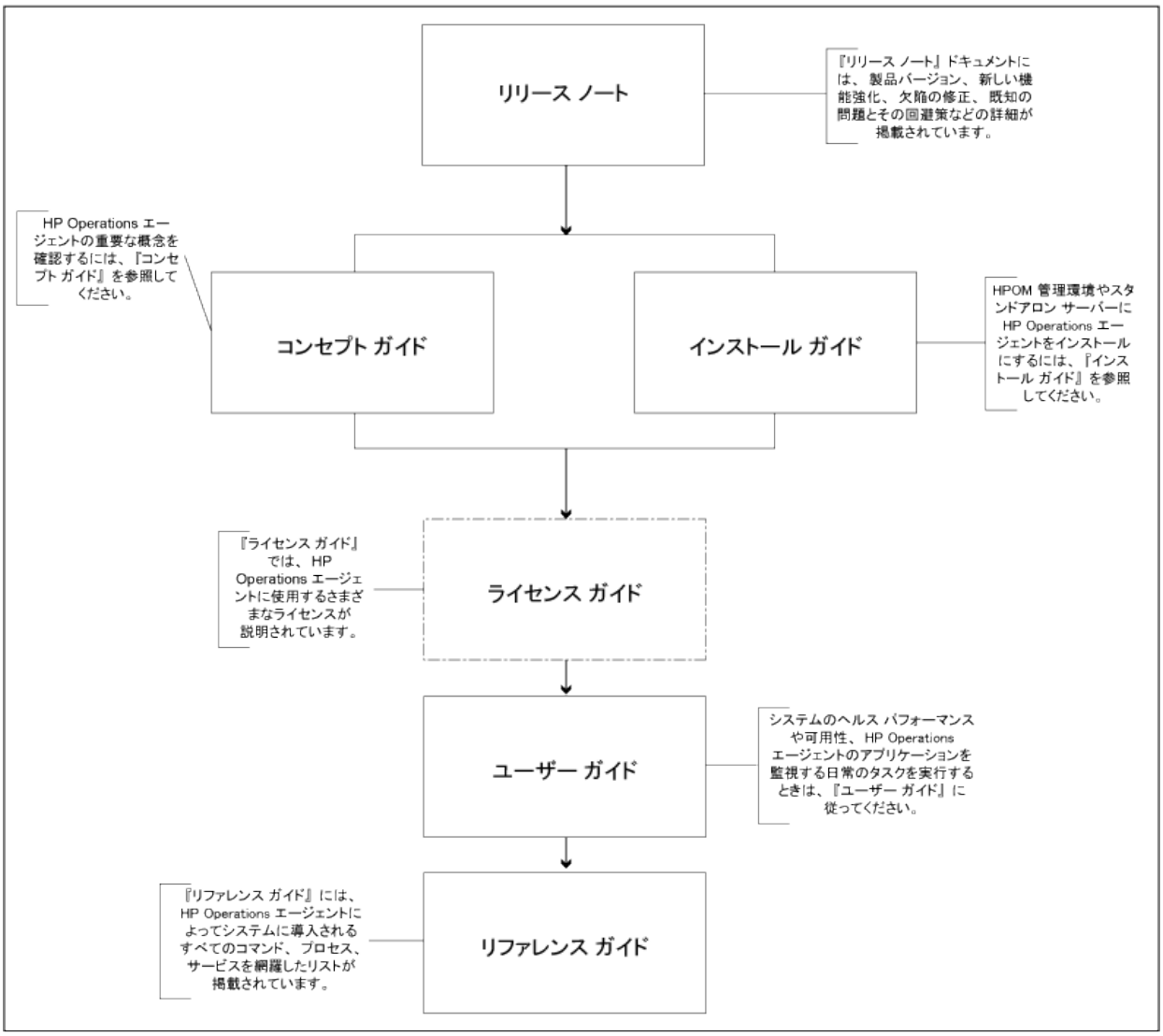
HP Operations エージェントでは、一連の設定変数を利用できます。これらの変数は、エージェント動作の制御に役立ちます。これらの変数には、`ovconfchg` コマンドを使って目的の値を設定できます。

このガイドには、HP Operations エージェントによってシステムに導入されるコマンド ライン ユーティリティ、サービス、プロセスに関する情報が記されています。また、HP Operations エージェントのデフォルト動作の設定に使用する各種設定変数についても説明します。

## ドキュメント マップ

ドキュメント マップには、HP Operations エージェントに関する主なドキュメントがすべて記載されています。このマップを利用して、必要なときに必要なドキュメントを見つけることができます。

**図 1: HP Operations エージェント** のドキュメント マップ


リリース ノート
『リリース ノート』ドキュメントには、製品バージョン、新しい機能強化、欠陥の修正、既知の問題とその回避策などの詳細が掲載されています。
HP Operations エージェントの重要な概念を確認するには、『コンセプト ガイド』を参照してください。
コンセプト ガイド
インストール ガイド
HPOM 管理環境やスタンドアロン サーバーに HP Operations エージェントをインストールにするには、『インストール ガイド』を参照してください。
『ライセンス ガイド』では、HP Operations エージェントに使用するさまざまなライセンスが説明されています。
ライセンス ガイド
ユーザー ガイド
システムのヘルス パフォーマンスや可用性、HP Operations エージェントのアプリケーションを監視する日常のタスクを実行するときは、『ユーザー ガイド』に従ってください。
『リファレンス ガイド』には、HP Operations エージェントによってシステムに導入されるすべてのコマンド、プロセス、サービスを網羅したリストが掲載されています。
リファレンス ガイド

# データ収集の管理

HP Operations エージェントには、監視対象システムのシステム パフォーマンス データを収集し記録するデータ コレクタが備わっています。データ収集プログラムの **scope** を使用することで、収集したデータをシステム上に格納できます。HP Performance Manager または HP Reporter を使用すれば、格納したデータの表示、分析が可能です。

scope コレクタでは、システム上で以下のタスクを実行できます。

- 監視対象システムのヘルスおよびパフォーマンスを示すメトリック データの収集
- 収集したメトリック データを異なるログ ファイルに記録

以下のツールを使用すると、これらのログ ファイルに記録されたデータを表示できます。

- extract および utility コマンド
- データ分析プログラム( HP Performance Manager など)

構成パラメータ ファイルの **parm** ファイルを使用すると、scope コレクタのデフォルトのデータ ログ作成機構を設定できます。**parm** ファイルのパラメータを変更することで、scope コレクタの以下のプロパティを制御できます。

- データ ログ作成間隔
- データ型
- ログ ファイルのサイズ

HP Operations エージェントをノードにインストールした後、**parm** ファイルを変更して scope のデータ収集機構を設定する必要があります。

## ログ ファイルベースのデータ ストアの使用

HP Operations エージェント の古いバージョン (バージョン 11.00 より古いもの) では、組み込みパフォーマンス コンポーネント (EPC) にデータを格納していました。HP Operations エージェント 11.00 (以降) では、ログ ファイルベースのデータ ストアにシステム パフォーマンス データが格納されます。ただし、EPC データ ストアは別の Smart Plug-in (SPI) でも使用できます。

SPI コレクタが EPC とログ ファイルベースのデータ ストアの両方の存在を検出すると、SPI が収集したデータは自動的にログ ファイルベースのデータ ストアに記録されます。

## ログ ファイルの収集

scope データ コレクタ (UNIX および Linux ノードでは scopeux、Windows ノードでは scopent) は、システム リソース利用率のパフォーマンス測定値を収集して要約し、**parm** ファイルの log line で指定されたデータ クラスに応じて、そのデータを次のログ ファイルに記録します。

logglob

logappl

logproc

logpcmd

logdev

logtran

logls

logindx

> **注 :** ログ ファイル内 のレコード のタイムスタンプは、データ収 集 の開 始 時 刻 を示 します。 対 象 プロセスの概 念 はフィルタリングです。 このフィルタリングは、ログに記 録 され、parm ファイルから制 御 されるデータの量 を最 小 化 するために役 立 ちます。 デフォルトでは scope は NFS データを記 録 しませんが、ローカル ファイル システム上 の GlancePlus を介 することで NFS データを表 示 できます。 NFS データのログを記 録 するように fstypes パラメータを設 定 できます (「fstypes」(38ページ)を参 照 )。

エージェント の旧 バージョンにロール バックすると、scope コレクタはログ ファイルへのログの記 録 を停 止することがあります。 この問 題 を解 決 するには、 以 下 の手 順 に従 います。

管 理 者 または root としてノード にログオンします。

scope コレクタを停 止 します。

*Windows の場 合 :*

**%ovinstalldir%bin\ovpacmd stop col**

*HP-UX、Linux、Solaris の場 合 :*

**/opt/perf/bin/ovpa stop scope**

*AIX の場 合 :*

**/usr/lpp/perf/bin/ovpa stop scope**

以 下 のディレクトリコマンド に移 動 します。

*Windows の場 合 :*

**%ovdatadir%datafiles**

*UNIX/Linux の場 合 :*

**/var/opt/perf/datafiles**

今 後 使 用 するためにすべてのログ ファイルのバックアップを作 成 し、ディレクトリのすべてのファイルを削 除 します。

バージョン 11.10 が実 行 されているシステムのバックアップ ログ ファイルを転 送 して、extract、export、utility コマンド を使 用 するか、データ分 析 ツール (HP Performance Manager など) を使 用 して、HP Operations エージェント のバージョン 11.10 で収 集 されたデータを表 示 できます。

ログ ファイルのバックアップを作 成 したら、**datafiles** ディレクトリのすべてのファイルを削 除 します。

scope コレクタを開 始 します。

*Windows の場 合 :*

**%ovinstalldir%bin\ovpacmd start col**

*HP-UX、Linux、Solaris の場 合*

**/opt/perf/bin/ovpa start scope**

*AIX の場合 :*

```
/usr/lpp/perf/bin/ovpa start scope
```

# logglob

**logglob** ファイルには、システム全体 (グローバル) のリソース利用率情報の測定値が記録されます。scope コレクタはグローバル データを要約し、**parm** ファイルに指定された間隔で、データを定期的に記録します。

# logappl

**logappl** ファイルには、**parm** ファイルに定義されたアプリケーションで実行されるプロセスの測定値の総計が記録されます。scope コレクタはアプリケーション データを要約し、**parm** ファイルに指定された間隔で、データを定期的に記録します。

# logproc

scope コレクタは対象と考えられるプロセスを特定し、特定されたプロセスの測定値の総計を **logproc** ファイルに記録します。scope は、以下の条件を基にプロセスを特定します。

プロセスの先頭

プロセスの末尾

**parm** ファイルで指定する構成の詳細

# logpcmd

**logpcmd** ファイルには、**logproc** ファイルに記録されるプロセスで実行される、コマンド ライン アクティビティの詳細が記述されています。

> **注 : logpcmd** ファイルのサイズ、ロールオーバー、記録間隔の制御はできません。

**logpcmd** ファイルは、ノードの以下のディレクトリに格納されます。

Windows の場合 :*%ovdatadir%***\datafiles**

UNIX (Linux) の場合 :**/var/opt/perf/datafiles**

ファイルに格納できるデータは最大 25 メガバイトです。データ収集機構が開始されると、scope コレクタが **logpcmd** ファイルの最初のインスタンスを作成します。このとき、末尾に 0 が付与されます。**logpcmd0** ファイルが最大サイズの 25 メガバイトに達すると、scope は **logpcmd** ファイルの 2 つめのインスタンスである、**logpcmd1** ファイルを作成します。

**logpcmd1** ファイルが最大サイズの 25 メガバイトを超えると、**logpcmd0** ファイルからデータのロールオーバーが開始されます。

## logdev

**logdev** ファイル。個 々 のデバイス パフォーマンスの測 定 値 が記 録 されます。scope コレクタはデバイス データを要 約 し、**parm** ファイルに指 定 された間 隔 で、データを定 期 的 に記 録 します。

## logtran

**logtran** ファイルには、ARM トランザクション データの測 定 値 が記 録 されます。scope コレクタはトラン ザクション データを要 約 し、**parm** ファイルに指 定 された間 隔 で、データを定 期 的 に記 録 します。トラン ザクション データの収 集 に関 する詳 細 は、「トランザクション追 跡 の紹 介 」を参 照 してください。

## logls

**logls** ファイルには、論 理 システムに関 する情 報 が記 録 されます。scope コレクタは論 理 システム デー タを要 約 し、**parm** ファイルに指 定 された間 隔 で、データを定 期 的 に記 録 します。

**logls** ファイルは、HPVM、Hyper-V Host、vSphere Management Assistant (vMA)、Solaris グローバ ル ゾーン、AIX-LPAR にある HP Operations エージェントでのみ使 用 可 能 です。

## logindx

**logindx** には、別 のログ ファイルのデータにアクセスするために必 要 な情 報 が記 録 されます。

# scope のステータス

scope の起 動 時 には、このログ ファイルの他 に 2 つのファイルが作 成 されます。1 つは **/var/opt/perf/datafiles/** ディレクトリにある **RUN** ファイルで、もう 1 つは **/var/opt/perf/** ディレクトリに ある **status.scope** ファイルです。

**RUN** ファイルは、scope プロセスが実 行 中 であることを示 すために作 成 されます。このファイルを削 除 すると、scope が終 了 します。

**/var/opt/perf/status.scope** ファイル は、scope プロセスのステータス ログまたはエラー ログとして使 用 できます。このファイルには、scope コレクタの開 始 時 と終 了 時 、または警 告 やエラーの発 生 時 に新 し い情 報 が追 加 されます。scope の最 新 のステータスおよびエラー情 報 を表 示 するには、perfstat -t コマ ンドを使 用 します。

# parm ファイル

**parm** ファイルは、scope に特 定 のパフォーマンス測 定 値 を記 録 させる命 令 を含 んだテキスト ファイル です。

新 規 インストールの際 には、HP Operations エージェントによってデフォルト の **parm** ファイルが 2 つの 異 なるディレクトリに作 成 されます。

Windows の場 合 :

> **注 :** Windows では、parm ファイルの拡 張 子 は .mwc (**parm.mwc**) となります。

*%ovinstalldir%***newconfig**

*%ovdatadir%*

HP-UX、Solaris、Linux の場合 :

**/opt/perf/newconfig**

**/var/opt/perf**

AIX の場合 :

**/usr/lpp/perf/newconfig**

**/var/opt/perf**

scope のデータ収集機構は、*%ovdatadir%* (Windows 版 ) または **/var/opt/perf** (UNIX、Linux 版 ) ディレクトリにある **parm** ファイルの設定により制御します。

デフォルトのデータ収集機構を変更する場合は、*%ovdatadir%* (Windows 版 ) または **/var/opt/perf** (UNIX、Linux 版 ) ディレクトリにある **parm** ファイルの設定を変更します。

ノード上の HP Operations エージェントを旧バージョンの HP Performance エージェント からアップグレードする場合は、**newconfig** ディレクトリの **parm** ファイルのコピーが、アップグレード プロセスによって更新されます。他のディレクトリにある **parm** ファイルは影響を受けず、引き続きノード上のデータ収集機構を管理します。つまり、この方法を用いることで、製品をアップグレードした後であっても、構成済みのデータ収集機構を保持できます。製品をアップグレードした後であれば、既存の **parm** ファイルの設定内容と、**newconfig** ディレクトリにある新しいバージョンの **parm** ファイルとを比較し、必要な変更を加えることができます。

**parm** ファイルは、平均的な量のログ ファイル データを収集するように設定されています。データの最大量は、システムによって異なります。また、「パラメータの説明」の size パラメータの説明も参照してください。

# parm ファイルの変更

**parm** ファイルは、ASCII フォーマットでファイルを保存できるワード プロセッサやエディタを使用して変更できます。

**parm** ファイルを変更または新規作成するときは、次の規則と表記規則が適用されます。

パラメータを指定すると、デフォルト値が無効になります。デフォルト値については、**newconfig** ディレクトリにある **parm** ファイルを参照してください。

**parm** ファイルにパラメータを指定する順序は、重要な意味を持ちません。

パラメータを複数回指定した場合、最後のパラメータのインスタンスが有効となります。

application 文を定義する file、user、group、cmd、argv1、or の各パラメータは、その application 文の後に指定する必要があります。

アプリケーションのパラメータは、プロセスが最初に適合するアプリケーションに総計されるように、順にリストで表示します。

すべてのコマンドおよびパラメータ文には、大文字、小文字、または大文字と小文字の組み合わせを使用できます。

空白またはカンマを使用して、各文に含まれるキーワードを区切ります。

**parm** ファイルのパラメータにはコメントを書き込むことができます。コメント コード (/*) またはポンド記号 (#) で始まる行は無視されます。

**parm** ファイルの変更後は、変更を有効にするために Performance Collection Component コンポーネントを再開始する必要があります。Performance Collection Component を再起動するには、次のコマンドを実行します。

*Windows の場合*

```
%ovinstalldir%bin\ovpacmd REFRESH COL
```

*HP-UX、Linux、Solaris の場合*

**/opt/perf/bin/ovpa -restart scope**

*AIX の場合*

```
/usr/lpp/perf/bin/ovpa -restart scope
```

リアルタイム メトリック アクセス (RTMA) コンポーネントを使用する場合は、perfd プロセスを再起動する必要があります。

*Windows の場合*

```
%ovinstalldir%bin\ovpacmd REFRESH RTMA
```

*HP-UX、Linux、Solaris の場合*

**/opt/perf/bin/pctl restart**

*AIX の場合 :*

```
/usr/lpp/perf/bin/pctl restart
```

## parm ファイルのパラメータ

scope は、収集パラメータ (**parm**) ファイル内の特定のパラメータによって制御されます。これらのパラメータは、次の定義と指定を行います。

scope 生ログファイルに対する最大ディスク スペース量の設定

記録するデータの種類の指定

データを記録する間隔の指定

記録するプロセスおよびメトリックの属性の指定

収集対象および記録対象のパフォーマンス データの種類の定義

ユーザーが定義可能な監視対象アプリケーションの指定。アプリケーションは、グループとして監視される 1 つまたは複数のプログラムになります。

scope が毎日のログファイルの保守アクティビティを実行する時間の指定。この指定は、システムの可用性に影響を与えないために行われます。

これらのパラメータを変更すると、監視対象システムの要件を満たすパフォーマンス データを scope で記録できます (「parm ファイルの変更」を参照してください)。

次の表の **parm** ファイル パラメータが scope で使用されます。これらのパラメータの中には、特定のシステム専用のものもあります (表を参照)。これらのパラメータの詳細は、「パラメータの説明」および「アプリケーション定義パラメータ」を参照してください。

**scope が使用する parm ファイル パラメータ**

| パラメータ | 値またはオプション |
|---|---|
| `id` | system ID |
| `log` | `global`<br><br>application [=prm] [=all]<br>([=prm] は HP-UX のみ)<br><br>process<br><br>device=disk,lvm,cpu,filesystem,all (lvm は HP-UX のみ)<br><br>transaction=correlator,resource (resource は HP-UX のみ)<br><br>logicalsystem<br>(Solaris では、論理システムは Solaris 10 以降のオペレーティング環境に対応)<br><br>AIX では、論理システムは AIX 5L V5.3 ML3 以降の LPAR、AIX 6.1 TL2 のグローバル環境の WPAR にのみ対応しています。<br><br>lpar ロギングを有効にする場合<br>logicalsystems=lpar<br>logicalsystems<br><br>wpar ロギングを有効にする場合<br>logicalsystems=wpar<br><br>lpar ロギングおよび wpar ロギングを有効にする場合<br>logicalsystems=lpar,wpar<br>logicalsystems=wpar,lpar<br>logicalsystems=all |
| `mainttime` | hh:mm (24 時間制) |
| `scopetransactions` | `onoff` |
| `subprocinterval` | 値は秒単位 (HP-UX は対象外) |
| `javaarg`<br><br>**メモ :**UNIX/Linux のみ | truefalse |
| `procthreshold` | cpu=*percent* |

| パラメータ | 値 またはオプション |
|---|---|
| (threshold と同じ) | disk=*rate* (Linux および Windows は対 象 外 )<br>memory=nn (値 はメガバイト単 位 )<br>nonewnokilledshortlived |
| appthreshold | cpu=*percent* |
| diskthreshold | util=*rate* |
| bynetifthreshold | iorate=*rate* |
| fsthreshold | util=*rate* |
| lvthreshold | iorate=*rate* |
| bycputhreshold | cpu=*percent* |
| fstypes | データの記 録 に特 定 のファイル システムのみを使 用 するには、構 文 *<file_system1>, <file_system2>, <file_system3>, ...* を使 用します。<br>ファイル システムを除 外 するには、構 文 !*<file_system>* を使 用します。 |
| wait | cpu=*percent*(HP-UX のみ)<br>disk=*percent*(HP-UX のみ)<br>mem=*percent*(HP-UX のみ)<br>sem=*percent*(HP-UX のみ)<br>lan=*percent*(HP-UX のみ) |
| application | *アプリケーション名* |
| file | *file name [, ...]* |
| argv1 | 最 初 のコマンド の引 数 [, ] |
| cmd | コマンド ラインの正 規 表 現 |
| user | *user login name [, ]* |
| group | *groupname [, ]* |
| or | |
| priority | 下 位 値 - 上 位 値<br>(範 囲 はプラットフォームによって異 なる) |
| size | (値 はメガバイト単 位 )<br>process=*nn* (最 大 値 は 4096)<br>下 位 のクラスの最 大 値 は 2048 です。<br>global=*nn* application=*nn*<br>device=*nn*<br>transaction=*nn*<br>logicalsystem=*nn* |

| パラメータ | 値 またはオプション |
|---|---|
| | |
| days | global=*nn* (値 は単 位 )<br>application=*nn*<br>process=*nn*<br>device=*nn*<br>transaction=*nn*<br>logicalsystem=*nn* |
| maintweekday | Sun\|Mon\|Tue\|Wed\|Thu\|Fri\|Sat |
| collectioninterval | process=ss （値は秒 単位）<br>global=ss |
| gapapp | blank<br>unassignedprocesses<br>existingapplicationname<br>other |
| flush | ss (値 は秒 単 位 )<br>0 (データ フラッシュを無 効 化 ) |
| zone_app<br>**メモ :** Solaris のみ | true<br>false<br>(Solaris 10 以降 のみ) |
| project_app<br>**メモ :** Solaris のみ | true<br>false<br>(Solaris 10 以降 のみ) |
| proccmd<br>**メモ :**UNIX/Linux のみ | 0 (プロセス コマンド のロギングを無 効 化 )<br>nnnn (プロセス コマンド の長 さの数 値 を参 照 。最 大 値 は 1024) |
| proclist<br>**メモ :** Solaris のみ | all (グローバル ゾーンの Performance Collection Component は、グローバル ゾーンおよび非 グローバル ゾーンで実 行 中 のすべてのプロセスを監 視 )<br>local (グローバル ゾーンの Performance Collection Component は、グローバル ゾーンで実 行 中 のプロセスのみを監 視 )<br>このパラメータは非 グローバル ゾーンでは無 効 となります。 |
| appproc<br>**メモ :** Solaris のみ | all (グローバル ゾーンおよび非 グローバル ゾーンに属 するアプリケーションのプロセスを使 用 して APP_ メトリックを計 算 するように Performance Collection Component を設 定 ) |

| パラメータ | 値 またはオプション |
|---|---|
| | local (グローバル ゾーンのみに属 するアプリケーションのプロセスを使 用し て APP_ メトリックを計 算 するように Performance Collection Component を設 定 )<br><br>このパラメータは非 グローバル ゾーンでは無 効 となります。 |
| `ignore_mt` | true (グローバル クラスの CPU メトリックは、システム内 のコアの有 効 数 に対 して正 規 化 した値 を報 告 )<br><br>false (グローバル クラスの CPU メトリックは、システム内 の CPU スレッド の有 効 数 に対 して正 規 化 した値 を報 告 )<br><br>`ineffective` （マルチスレッド を無 効 化 ）<br><br>**メモ :**このパラメータは HP-UX では無 効 となります。上 記 のモード を切 り 替 えるには、HP-UX で **midaemon -ignore_mt** コマンド を実 行 する必 要 があります。詳 細 情 報 は、「コアに基 づいた正 規 化 によって計 算 され たメトリックの記 録 」を参 照 してください。 |
| `cachemem`<br><br>**メモ :**Linux のみ | f または free (GBL_MEM_UTIL メトリックの値 を計 算 する場 合 、HP Operations エージェント はバッファ キャッシュ サイズを含 めません)<br><br>u または user (GBL_MEM_UTIL メトリックの値 を計 算 する場 合 、HP Operations エージェント はバッファ キャッシュ サイズを含 めます) |

## パラメータの説 明

ここでは、**parm** ファイルの各 パラメータについて説 明します。

ID

Log

Thresholds

scopetransactions

subprocinterval

gapapp

fstypes

wait

size

mainttime

days

maintweekday

javaarg

Flush

zone_app

project_app

proclist

appproc

proccmd

ignore_mt

cachemem

## ID

システム ID 値 は、システムを識 別 する文 字 列 です。割り当 てられるデフォルト ID はシステムのホスト名 です。割り当 てられるデフォルト ID を変 更 する場 合 は、すべてのシステムが一 意 の ID 文 字 列 を持つことを確 認してください。この識 別 子 は、データを収 集したシステムを示 すためにログ ファイルに収 められます。最 大 39 文 字 を指 定 できます。

## log

log パラメータは、scope で収 集 するデータ型 を指 定 します。

log global を指 定 するとscope で **logglob** ファイルにグローバル レコードを記 録 できます。システムのパフォーマンス データを表 示 し、分 析 するには、グローバル データ レコード が必 要 です。グローバル メトリックは、アプリケーション データおよびプロセス データのログ オプションや値 による影 響 は受 けません。

log application を指 定 するとscope で **logappl** ファイルにアクティブなアプリケーションのレコードを記 録できます。デフォルト では、インターバル中 にアクティブなプロセスがあったアプリケーションのみが記 録 されます。

parm ファイルで **log application=all** を指 定 すると、アプリケーションがアクティブかどうかに関 係 なく、すべてのアプリケーションが scope によってインターバルごとに **logappl** ファイルに記 録 されます。

application=all オプションは、アプリケーション アラームの使 用 と関 連 がある特 定 の状 況 に適 しています。たとえば、アプリケーションがアクティブでなくなった場 合 にアラームを発 生 させることができます (APP_ALIVE_PROC)。

このオプションを有 効 にすると、すべてのアプリケーションがインターバルごとに記 録 されるため、**logappl** のサイズがより短 時 間 で大 きくなります。**utility** プログラムの scan 関 数 を使 用 すると、scope ログ ファイルの利 用 率 を監 視 できます。

log application=prm パラメータを指 定 すると、scope により、アクティブな Process Resource Manager (PRM) グループを **logappl** ファイルに書 き込 むことができます (HP-UX のみ)。このパラメータを指 定 すると、scope は **parm** ファイルにリストされているユーザー定 義 のアプリケーション セットを記 録しません。また、収 集 されたすべてのアプリケーション メトリックは、PRM コンテキストを反 映 し、APP_NAME_PRM_GROUPNAME メトリックによってグループ化 されます。

アプリケーションのログ オプションは、グローバル データおよびプロセス データには影 響 しません。

**log process** を指 定 するとscope で **logproc** ファイルに対 象 プロセスに関 する情 報 を記 録 できます。ログの対 象 となるのは、プロセスが最 初 に作 成 されたとき、終 了 したとき、および parm ファイルで定 義されたアプリケーションのしきい値 を超 えたときです。プロセスのしきい値 のログ オプションは、グローバル データおよびアプリケーション データには影 響 しません。

log device=disk,lvm,cpu,filesystem を指定すると、scope によって、各ディスク、論理ボリューム (HP-UX のみ)、CPU、ファイルシステムに関する情報が **logdev** ファイルに記録されます。

> **注 :** 監視対象のシステムが HP-UX オペレーティングシステムで実行されていない場合は、lvm を使用しないでください。

デフォルトでは、インターバル中に I/O が発生したディスク、ボリューム、およびインターフェイスのみが記録されます。netif (論理 LAN デバイス) レコードおよびディスクレコード (HP-UX) は、選択したログデバイスオプションに関係なく、常に記録されます。

たとえば、各ディスク、論理ボリューム、CPU、ネットワークインターフェイスのレコードは記録するが、各ファイルシステムは記録しない場合、以下の設定を使用します。

log device=disk,lvm,cpu

filesystem を指定すると、マウントされているすべてのローカルファイルシステムが、アクティビティに関係なくすべてのインターバルごとに記録されます。

**parm** ファイルで log device=all を指定すると、scope によって、すべてのディスク、論理ボリューム、CPU、ネットワークインターフェイスが、それらのデバイスがアクティブかどうかに関係なく、インターバルごとに **logdev** ファイルに記録されます。

このオプションを有効にすると、すべてのデバイスがインターバルごとに記録されるため、**logdev** のサイズがより短時間で大きくなります。ログファイルの利用率とサイズをモニタするには、**utility** プログラムの scan 関数を使用します。

log transaction を指定すると scope で **logtran** ファイルに ARM トランザクションレコードを記録できます。scope でデータを収集するには、アプリケーション応答測定 (ARM:Application Response Measurement) API が組み込まれているプロセスを実行する必要があります(詳細については、What is Transaction Tracking? on page 335を参照してください)。

log transaction パラメータのデフォルト値は、no resource と no correlator です。

リソースデータ (HP-UX のみ) または相関データを収集するには、**log transaction=resource** または **log transaction=correlator** を指定します。両方のデータを記録するには、 **log transaction=resource, correlator** を指定します。

log logicalsystems を指定すると scope で **logls** ファイルに論理システムに関する情報を記録できます。論理システムに関するデータは、**parm** ファイルで指定した間隔で定期的に要約されます。

AIX 6.1 TL2 での LPAR および WPAR の BYLS ロギングは、**parm** ファイルの logicalsystems パラメータを使用して設定できます。「scope が使用する parm ファイル パラメータ」(30ページ)を参照してください。

ログオプションに関係なく、自動的にログファイルが作成されます。特定の種類の記録が無効になっていても、対応するログファイルが監視対象システムから自動的に削除されることはありません。

オプションなしで log を指定すると、scope はグローバルデータとプロセスデータのみを記録します。

## Thresholds

しきい値パラメータを指定すると、scope は、システムに関する不必要で、重要度の低い詳細情報をフィルタリングし、重要な情報のみを記録します。

それぞれのメトリックのクラスのしきい値の指定には、以下のパラメータを使用します。データのクラスの特定のインスタンスに関して、指定したしきい値を超えた場合は、scope によってそのインスタンスに関するレコードが記録されます。

しきい値に小さな値を指定すると、scope でより多くのデータが記録され、逆にしきい値に大きな値を指定すると、scope で記録されるデータが少なくなるため、記録されるレコードの数が概ね少なくなります。使用可能なしきい値のパラメータのリストを次に示します。

「procthreshold」(36ページ)

appthreshol

diskthreshold

bynetifthreshold

fsthreshold

lvthreshold

bycputhreshold

## procthreshold

procthreshold パラメータは、対象プロセスの条件を指定するアクティビティレベルを設定するために使用します。このパラメータを使用するには、プロセスの記録を有効にする必要があります。procthreshold は、記録されるプロセスのみに影響するもので、データの他のクラスには影響しません。

しきい値のオプションは、(カンマで区切って) 同じパラメータ行に指定します。

プロセス データ用の procthreshold オプション

| | |
|---|---|
| cpu | CPU の利用率を設定します。この利用率を超えたプロセスが対象プロセスとなり、記録されます。<br><br>パーセント値は、全体的な CPU の使用率を示す実数です。たとえば、cpu=7.5 は、1 分間のサンプルで CPU の利用率が 7.5% を超えたプロセスが記録されることを示します。 |
| disk | (Linux および Windows では使用できません)物理的なディスク I/O の 1 秒あたりの回数を設定します。この回数を超えたプロセスが対象プロセスとなり、記録されます。<br><br>この値は実数です。たとえば、disk=8.0 を指定すると、1 秒あたりの平均物理ディスク I/O 率が 8 KB を超えたプロセスが記録されます。 |
| memory | メモリのしきい値を設定します。このしきい値を超えたプロセスが「対象」プロセスとなり、記録されます。<br><br>値はメガバイト単位で、100 KB の精度で最も近い値を指定します。メモリのしきい値が設定されている場合は、PROC_MEM_VIRT メトリックの値と比較されます。メモリのしきい値を超えた各プロセスは、ディスクおよび CPU のプロセス ログしきい値の場合と同様に記録されます。 |
| nonew | 新しいプロセスがしきい値を超えなかった場合に、そのプロセスの記録を停止します。指定されていない場合は、新しいプロセスはすべて記録されます。HP-UX で |

| | |
|---|---|
| | 、1 秒 以 上 の新しいプロセスのみが記 録 されます。 |
| nokilled | 終 了 したプロセスがしきい値 を超 えなかった場 合 に、そのプロセスの記 録 を停 止 します。指 定 されていない場 合 は、すべての Kill (終 了 ) されたプロセスが記 録 されます。HP-UX では、shortlived が指 定 されていない場 合 は、1 秒 以 上 の Kill されたプロセスのみが記 録 されます。 |
| shortlived | インターバル内 での実 行 時 間 が 1 秒 未 満 のプロセスが記 録 されます(多 くの場 合 、記 録 されるプロセス数 が大 幅 に増 加 します)。scope によって **parm** ファイル内 で threshold shortlived が検 出 されると、nonew および nokilled オプションが削 除 されている場 合 は、cpu または disk threshold の指 定 に関 係 なく、**shortlived** に該 当 するプロセスが記 録 されます。デフォルトでは、shortlived プロセスは記 録 されません(shortlived プロセスを記 録 する必 要 がない場 合 は、threshold パラメータで shortlived を指 定 しないようにしてください)。 |
| process | procthreshold は、PROCESS クラスのしきい値 を指 定 します。デフォルト値 は以 下 のようになります。<br>最 後 の時 間 間 隔 内 の CPU のプロセッサ使 用 率 が 10% を超 えているプロセス<br>900 MB を超 える仮 想 メモリが設 定 されているプロセス<br>1 秒 あたりの平 均 物 理 ディスク I/O 率 が 5 KB を超 えるプロセス |

## subprocinterval

subprocinterval パラメータを指 定 すると、scope がプロセス データのサンプリングに使 用 するデフォルトの設 定 が変 更 されます。プロセス データおよびグローバル データは、parm ファイルで指 定 した間 隔 で定 期 的 に記 録 されます。ただし、scope は、短 期 間 のアクティビティを対 象 とするため、数 秒 ごとに測 定 します。デフォルトでは、このインストルメンテーションのサンプリング インターバルは 5 秒 です。プロセス データのログ間 隔 は、subprocinterval の偶 数 倍 になります。詳 細 情 報 は、「データ記 録 間 隔 の設 定 」(43ページ)を参 照 してください。

無 数 のアクティブなスレッド またはプロセスがあるシステムでは、scope の全 体 的 なオーバーヘッドを減 らすために、subprocinterval を長 くする必 要 も生 じます。また、多 数 の短 期 間 のプロセスを記 録 する必 要 があるシステムでは、subprocinterval を短 く設 定 することもできますが、この場 合 は scope のオーバーヘッド への影 響 を注 意 して監 視 しなければなりません。この設 定 には、**parm** ファイルに指 定 したプロセスのログのインターバルの因 数 となる値 を指 定 する必 要 があります。

| | |
|---|---|
| | subprocinterval に小 さな値 を指 定 すると、グローバル メトリックとアプリケーションの合 計 とのギャップが減 少 します (HP-UX 以 外 の他 のすべてのオペレーティング システムで有 効 )。 |

## gapapp

parm ファイルの gapapp パラメータは、データのアプリケーション クラスの変 更 を制 御 することで、グローバル (システム全 体 ) データとアプリケーション データの合 計 との差 異 を説 明 します。

アプリケーション データは、プロセス レベルのインストルメンテーションによって得 られます。通 常 、グローバル メトリックとアプリケーションの合 計 には差 異 が生 じます。プロセス作 成 の割 合 の高 いシステムでは、その差 異 が大 きくなります。以 下 のオプションを選 択 できます。

gapapp が空白。gapapp という名前のアプリケーションがアプリケーション リストに追加されます。

gapapp = UnassignedProcesses。UnassignedProcesses という名前のアプリケーションがアプリケーション リストに追加されます。

gapapp = ExistingApplicationName (または) gapapp = other。指定したアプリケーションにグローバル値との差異が追加されます。個別に記録されたり、新しいエントリがアプリケーション リストに追加されることはありません。

## fstypes

fstypes パラメータを使うと、システムにある特定のファイル システムを監視できます。デフォルトでは、scope コレクタは、システムに接続されているすべてのファイル システムのデータのログを記録します。fstypes パラメータを指定すると、特定のファイル システムだけのデータ収集を有効にできます。

**parm** ファイルでは、監視するファイル システムの名前に fstypes パラメータを設定する必要があります。カンマで区切ることで、複数のファイル システム名を指定できます。

このパラメータの構文は次のようになります。

fstypes=*<file_system1>*,*<file_system2>*, ...

*<file_system1>* はファイル システム タイプです。

HP-UX、Linux、AIX の場合、利用可能なファイル システムとファイル システム タイプのリストは、**/etc/fstab** ファイルにあります。

AIX の場合、利用可能なファイル システムとファイル システム タイプのリストは、**etc/filesystems** ファイルにあります。

Solaris の場合、利用可能なファイル システムとファイル システム タイプのリストは、**/etc/vfstab** ファイルにあります。

Windows でファイル システム タイプを確認するには、Windows Explorer でディスク ドライブを右クリックして、[**プロパティ**] をクリックします。表示されるファイル システム値は、fstypes パラメータで指定できるファイル システム タイプです。

このパラメータには包含リストや除外リストを設定できます。ファイル システム タイプをカンマ区切りでリストにすると包含リストになります。このリストを使って、指定したファイル システムのデータだけを監視するようにエージェントを設定できます。

たとえば、次のようになります。

fstypes = tmpfs, mvfs, nfs

fstypesパラメータの設定中、オペレーティング システムのコマンドから返されるファイル システム名を使用します。

このパラメータでは除外リストも設定できます。これは、! の文字で始まるファイル システム タイプのカンマ区切りリストです。! の文字をリストのはじめに指定すると、そのリストに属するすべてのファイル システムを監視しないようにエージェントを設定できます。

たとえば、次のようになります。

fstypes = !tmpfs, mvfs

上記の設定により、すべての一時ファイル システムおよび監視対象ノードに接続されている複数バージョンのファイル システムの監視が停止します。

nfs および nfs3 の両方のファイルシステムを監視する場合、fstypes = nfs, nfs3 ではなくfstypes = nfs* と指定できます。

## wait

wait パラメータ (HP-UX のみ) を使用すると、システムリソースを待っているプロセスの詳細を取得できます。wait パラメータの値はパーセンテージで指定できます。プロセスがシステムリソース (cpu、disk、mem、sem、lan) を待つ場合。待ち間隔の割合が wait パラメータに指定された値を上回る場合に、プロセスの詳細が **logproc** ファイルに記録されます。

値およびオプションについては「scope が使用する parm ファイルパラメータ」(30ページ)を参照してください。

たとえば、プロセスの記録間隔が 60 秒に、CPU の wait パラメータが 50% に定義されている場合、CPU を待っている時間が 30 秒以上となったすべてのプロセスが **logproc** ファイルに記録されます。

## size

size パラメータは、生ログファイルの最大サイズを (メガバイト単位で) 設定するために使用されます。サイズは 1 メガバイト未満には設定できません。

scope コレクタは、開始時にこれらの指定を読み込みます。これらのログファイルのいずれかが収集作業中に最大サイズに達した場合でも、そのログファイルは、自動的にロールバックされる mainttime 時までは増大し続けます。ロールバック中、最も古いものから 25% のデータがログファイルから削除されます。`size` パラメータによって制限されていない場合でも、生ログファイルには最大 1 年分のデータしか保存されません。「Utility Scan Report Details」の「Log File Contents Summary 」および「Log File Empty Space Summary 」を参照してください。

**parm** ファイル内の size 指定を変更すると、scope の開始時にその変更が検出されます。最大ログファイルサイズが、既存のデータを収められないほど小さくなると、scope の開始時にサイズ変更が自動的に行われます。新たに指定した最大サイズで既存のデータを収められる場合、どのような動作も行われません。

resize コマンドは、元のログファイルを削除する前に、TMPDIR 環境変数で設定されているディレクトリに新しいファイル scopelog を作成します。「resize」の使い方のセクションを参照してください。

ログファイルの最大サイズについて行った変更は、mainttime パラメータで指定した時刻に有効になります。

> **注 :** size パラメータに関係なく、scope ログファイルの最大サイズも、過去 1 年間に格納されたデータの総量に限定されます。生 scope ログファイルには 1 年を超えるデータを記録することができず、ログがその期間を超えた場合、1 年を超える古いデータが上書きされます。1 年を超えるデータが必要な場合のアーカイブログファイルの作成方法については、extract を参照してください。

## mainttime

ログファイルは、必要に応じて毎日特定の時刻に scope によってロールバックされます。デフォルトの時刻は、mainttime パラメータを使用して変更できます。たとえば、**mainttime=8:30** を設定すると、ログファイルの保守は毎日 8:30 am に行われます。

mainttime は、システム利用率が最も低い時刻に設定することをお勧めします。

> **注 :** ログファイルの保守では、1 日を超える古いデータのみがロールアウトされますので、**logproc** のようにすぐに大きくなって 24 時間以内に限度に達するようなログファイルの場合は、設定した最大サイズを超えるおそれがあります。

## days

days パラメータは、すべての生データログファイルが格納可能なデータの最大日数を指定します。このパラメータの値は、1 から 365 の範囲で指定します。このパラメータを使用することで、scope データコレクタでログファイルの保守を行えます。

データ収集時に、ログファイル内のデータ日数が days パラメータで指定した日数に達した場合、maintweekday パラメータで指定した日に一致するまでデータ収集が続行されます。maintweekday に達すると、ログファイルは maintime に自動的にロールバックされます。ロールバック中、days パラメータが最大値に達した後に収集されたデータがログファイルから削除されます。

> **注 :** データ収集中にログファイルのロールバックが行われる際、days パラメータより前の特定の日に size パラメータに指定した値になった場合、days パラメータよりも size パラメータが優先されます。

たとえば、**parm** ファイルに「size global=20」および「days global=40」と設定されており、ログファイルがデータがログファイルに記録される 40 日前に最大サイズの 20 MB に達した場合、size パラメータに基づいてログファイルのロールバックが実行されます。

## javaarg

> **注 :** このパラメータは UNIX/Linux のみで有効です。

javaarg パラメータは、true または false のいずれかに設定するフラグです。proc_proc_argv1 メトリックの値にのみ影響します。

javaarg を false に設定する、または **parm** ファイルに指定しない場合、**proc_proc_argv1** メトリックは、常にプロセスのコマンド文字列の最初の引数の値に設定されます。

javaarg が true に設定されていると、コマンド文字列に class または jar 仕様があれば、proc_proc_argv1 メトリックよりも優先されます (Java プロセスの場合のみ)。つまり、オペレーティングシステムの提供する引数リストにデータが存在すると仮定した場合、ファイル名が java または jre のプロセスについては、proc_proc_argv1 メトリックよりも、-classpath または -cp の後にない最初の引数 (先頭にダッシュのないもの) が優先されます。

複雑に思えるかもしれませんが、Java プロセスがシステムで実行されている場合は非常に簡単です。つまり、**javaarg=true** と設定すると、proc_proc_argv1 メトリックが class または jar 名と共に記録されます。これは Java 固有のアプリケーションを定義する場合に便利です。クラス名が proc_proc_argv1 にあれば、argv1= application 修飾子を使用して、クラス名でアプリケーションを定義できます。

## Flush

flush パラメータは、アプリケーション データおよびデバイス データのすべてのインスタンスを記録する、データの記録間隔 (秒単位) を指定します。flush 間隔は、300 から 32700 の範囲にある 300 の偶数倍の数字で指定します。

アプリケーション データおよびデバイス データのすべてのインスタンスに対する、flush 間隔のデフォルト値は 3600 秒です。

flush パラメータを無効にするには、その値に 0 (ゼロ) を指定します。flush パラメータが 0 に設定されていると、scope は **parm** ファイルで指定されたしきい値に一致しないアプリケーション データおよびデバイス データを記録しません。

## zone_app

zone_app フラグを使用すると、Operations エージェントで Solaris ゾーン固有のデータを収集できます。zone_app フラグが true に設定されていると、すべてのアプリケーション クラス (APP_*) メトリックの収集に影響します。**parm** ファイルにリストされているすべてのユーザー定義アプリケーション セットは無視され、Operations エージェントがインストールされているマシンで実行されているゾーンに基づいて、アプリケーション メトリックが収集されます。

たとえば、2 つの非グローバルゾーン「zone1」および「zone2」を持つ Solaris マシンを考えた場合、Operations エージェントは parm ファイルのアプリケーション セットを無視し、「global」、「zone1」、「zone2」という名前の 3 つのアプリケーションを作成します。各アプリケーションのパフォーマンス測定は、それぞれのゾーンで実行されているプロセスから取得された測定値を基準にします。たとえば、「zone1」ゾーンの APP_CPU_TOTAL_UTIL メトリックは、zone1 で実行されているすべてのプロセスに対する CPU 使用率の値に基づいて計算されます。

zone_app フラグが無効に設定されているか、Operations エージェントが Solaris 9 以前のバージョンで実行されている場合、APP_LS_ID メトリックは使用不可 (na) の値を報告します。アプリケーションのグループ化は **parm** ファイルにリストされているユーザー定義のアプリケーション セットを基準に実行されます。

**注:** ゾーンは、Solaris 10 以降のバージョンのみが対応しています。

## project_app

このパラメータを true に設定すると、Performance Collection Component は各 Solaris プロジェクトをアプリケーションとして見なします (プロジェクト ID をアプリケーション ID と見なします)。Solaris プロジェクトを無視するには、このパラメータを false に設定します。

| | |
|---|---|
| | このパラメータは、Solaris 10 以降のバージョンでのみサポートされています。 |

## proclist

このパラメータは Solaris グローバルゾーンでのみ使用できます。グローバルゾーン以外で実行されているエージェントには影響しません。

グローバルゾーンでこのパラメータを all に設定すると、Operations エージェントはグローバルゾーンとグローバルゾーン以外にあるすべてのプロセスを監視します。グローバルゾーンに属するプロセスのみを監視するには、このパラメータを local に設定します。

## appproc

このパラメータは、Solaris でのみ使用できます。このパラメータはグローバルゾーンでのみ使用できます。グローバルゾーン以外で実行されているエージェントには影響しません。

グローバルゾーンでこのパラメータを all に設定すると、すべての APP_ メトリックの値を計算しているときに、Performance Collection Componentはグローバルゾーンとグローバルゾーン以外のアプリケーションのプロセスを含めます。APP_ メトリックの計算にグローバルゾーンのアプリケーションだけを含めるには、このパラメータを local に設定します。

## proccmd

**注 :** このパラメータは UNIX/Linux のみで有効です。

proccmd パラメータを使用すると、HP Operations エージェントデータストアにプロセスコマンドを記録できます。プロセスコマンドの長さは、パラメータ内に数値で指定します。最大値は 1024 です。デフォルトでは、このパラメータの値は 0 に設定されており、プロセスコマンドの記録は無効になっています。

## ignore_mt

このパラメータを true に設定すると、Performance Collection Component は、監視対象システム上のアクティブなコアの数に対してメトリック値を正規化した後に、Global クラスの CPU に関連するすべてのメトリックを記録します。

このパラメータを false に設定すると、Performance Collection Component は、監視対象システム上のスレッドの数に対してメトリック値を正規化した後に、Global クラスの CPU に関連するすべてのメトリックを記録します。

このパラメータは HP-UX では無効となります。上記のモードを切り替えるには、HP-UX で **midaemon -ignore_mt** コマンドを実行する必要があります。詳細情報は、「コアに基づいた正規化によって計算されたメトリックの記録」(47ページ)を参照してください。

システムのマルチスレッド プロパティが無効になっている場合、このパラメータは Performance Collection Component によって無視されます。その結果、GBL_IGNORE_MT メトリックの値は true として記録されます。

**注 :** Windows、Linux、Solaris システムで、同時マルチスレッディング (Simultaneous Multithreading (SMT)) を有効/無効にした場合は、システムを再起動する必要があります。

## cachemem

**parm** ファイルの cachemem パラメータを使うと、メモリ使用率合計のデータ (GBL_MEM_UTIL メトリック値 ) をレポートするときに、バッファ キャッシュ サイズを含めるようにエージェントを設定できます。このパ

ラメータは Linux にのみ有効です。

デフォルトでは、このパラメータは free (f) に設定されています。これは、エージェントがバッファ キャッシュ サイズを GBL_MEM_UTIL メトリック値に含めないことを示します。

バッファ キャッシュ サイズを GBL_MEM_UTIL メトリック値に含めるには、このパラメータを user (u) に設定する必要があります。

## アプリケーション定義パラメータ

application、file、user、group、cmd、argv1、or の各パラメータは、アプリケーションの定義に関するものです。

Performance Collection Component では、論理的に関連性のあるプロセスを 1 つのアプリケーションにまとめることで、そのプロセスがメモリや CPU などのコンピュータ リソースに与えた総合的な影響を記録できます。

> **注 :** PRM モード (HP-UX のみ) では、アクティブな PRM グループが記録され、**parm** ファイルにリストされているユーザー定義のアプリケーション セットは無視されます。

アプリケーションは、ファイルのリスト (基本プログラム名 )、コマンドのリスト、またはファイルおよびコマンドの組み合わせで設定できます。ユーザー名、グループ名または引数選択で修飾することも可能です。これらのすべてのアプリケーション修飾子は個別で使用できるほか、それぞれを組み合わせて使用することもできます。たとえば、cmd および user の 2 つの修飾子が使用されている場合、プロセスを該当のアプリケーションに含めるには、プロセスが両方のコマンド文字列およびユーザー名の仕様を満たす必要があります。次に、各修飾子の詳細について説明します。

> **注 :** システム上の各プロセスは、1 つのアプリケーションにのみ属します。複数のアプリケーションでカウントされるプロセスはありません。

## データ記録間隔の設定

scope が使用するデフォルトの収集間隔は、プロセス データの場合が 60 秒、グローバル データおよびデータの他のすべてのクラスの場合が 300 秒です。**parm** ファイルの collectioninterval パラメータを使用するとこのパラメータを無効にできます。値は以下の条件を満たす必要があります。

プロセス データの収集間隔は、5 から 60 秒で設定します (5 秒間隔 )。プロセス データの収集間隔は、subproc 間隔 (「subprocinterval」を参照 ) の倍数とし、グローバル収集間隔に等分する必要があります。

グローバル データの収集間隔は、15、30、60、300 秒のいずれかで設定します。グローバル収集間隔はプロセス間隔以上の数値とし、プロセス収集間隔の倍数に設定する必要があります。グローバル収集間隔は、グローバル メトリックおよび、ファイルシステムやアプリケーションといったすべての非プロセス メトリックに適用されます。

# vMA ノードでのデータ収集の設定

HP Operations エージェントでは `viserver` デーモンを使用して、vMA システムにデータのログを記録します。`/var/opt/perf` にある次の構成ファイルで`viserver` 設定を構成します。

`viserver.properties`

```
VILog4j.xml
```

# ハイパースレッディング/マルチスレッディングが有効なシステムにおける CPU メトリックの正規化

ハイパースレッディング/マルチスレッディング (HT/SMT) が有効なシステムでは、物理的 CPU が2つ以上のハードウェアスレッドに対応しています。結果として、複数のソフトウェアプロセスまたはソフトウェアスレッドが、ハードウェアスレッド上で同時に実行されます。マルチコアプロセッサを持つシステムでは、複数のスレッドを個々のコアで同時に実行できます。

Performance Collection Component には複数の CPU 関連メトリックがあり、監視対象システムの CPU 使用率の分析および把握が可能です。デフォルトでは、HT/SMT が有効になっているすべてのシステムで、Performance Collection Component は収集したデータを監視対象システム上で利用可能なスレッドの数に対して正規化し、CPU 関連のすべてのメトリックの値を計算します。単一のスレッドが CPU コア全体を完全に使用している場合、スレッドを基にした正規化によって計算された値が、CPU 使用率の実状を表さない場合があります。

本バージョンの HP Operations エージェントには新しい構成パラメータ ignore_mt が導入されています。これを使用すると、Performance Collection Component で、コアに基づいた正規化により計算された CPU 関連のデータを記録するように設定できます。コアに基づいた正規化によるメトリック値は CPU 使用率の状態をより正確に表しているため、システムのパフォーマンスを分析する際により有効な決定を行うための手助けとなります。

## メトリックのリスト

次の表には、スレッドやコアの数に対して正規化されるすべてのプラットフォームの CPU 関連のメトリックのリストが記されています。

**アクティブなコアやスレッドの数に対して正規化されるメトリックのリスト**

| メトリック | Linux | HP-UX | Solaris | AIX | Windows |
|---|---|---|---|---|---|
| GBL_CPU_TOTAL_UTIL | Y | Y | Y | Y | Y |
| GBL_CPU_TOTAL_UTIL_CUM | Y | Y | Y | Y | なし |
| GBL_CPU_TOTAL_UTIL_HIGH | Y | Y | Y | Y | なし |
| GBL_CPU_TOTAL_TIME | Y | Y | Y | Y | Y |
| GBL_CPU_TOTAL_TIME_CUM | Y | Y | Y | Y | なし |
| GBL_CPU_SYS_MODE_UTIL | Y | Y | Y | Y | Y |
| GBL_CPU_SYS_MODE_UTIL_CUM | Y | Y | Y | Y | なし |
| GBL_CPU_SYS_MODE_UTIL_HIGH | Y | なし | Y | Y | なし |
| GBL_CPU_SYS_MODE_TIME | Y | Y | Y | Y | Y |
| GBL_CPU_SYS_MODE_TIME_CUM | Y | Y | Y | Y | なし |

| メトリック | Linux | HP-UX | Solaris | AIX | Windows |
|---|---|---|---|---|---|
| GBL_CPU_USER_MODE_TIME | Y | Y | Y | Y | Y |
| GBL_CPU_USER_MODE_TIME_CUM | Y | Y | Y | Y | なし |
| GBL_CPU_USER_MODE_UTIL | Y | Y | Y | Y | Y |
| GBL_CPU_USER_MODE_UTIL_CUM | Y | Y | Y | Y | なし |
| GBL_CPU_USER_MODE_UTIL_ HIGH | Y | なし | Y | Y | なし |
| GBL_CPU_NICE_UTIL | Y | Y | なし | なし | なし |
| GBL_CPU_NICE_UTIL_CUM | Y | Y | なし | なし | なし |
| GBL_CPU_NICE_UTIL_HIGH | Y | Y | なし | なし | なし |
| GBL_CPU_NICE_TIME | Y | Y | なし | なし | なし |
| GBL_CPU_NICE_TIME_CUM | Y | Y | なし | なし | なし |
| GBL_CPU_INTERRUPT_UTIL | Y | Y | なし | なし | Y |
| GBL_CPU_INTERRUPT_UTIL_CUM | Y | Y | なし | なし | なし |
| GBL_CPU_INTERRUPT_UTIL_HIGH | Y | Y | なし | なし | なし |
| GBL_CPU_INTERRUPT_TIME | Y | Y | なし | なし | Y |
| GBL_CPU_INTERRUPT_TIME_CUM | Y | Y | なし | なし | なし |
| GBL_CPU_IDLE_UTIL | Y | Y | Y | Y | Y |
| GBL_CPU_IDLE_UTIL_CUM | Y | Y | Y | Y | なし |
| GBL_CPU_IDLE_UTIL_HIGH | Y | Y | Y | Y | なし |
| GBL_CPU_IDLE_TIME | Y | Y | Y | Y | Y |
| GBL_CPU_IDLE_TIME_CUM | Y | Y | Y | Y | なし |
| GBL_CPU_WAIT_UTIL | Y | Y | Y | Y | なし |
| GBL_CPU_WAIT_TIME | Y | なし | Y | Y | なし |
| APP_CPU_NICE_TIME | なし | Y | なし | なし | なし |
| APP_CPU_NICE_UTIL | N | Y | N | N | N |
| APP_CPU_NORMAL_TIME | N | Y | N | N | N |
| APP_CPU_NORMAL_UTIL | N | Y | N | N | N |
| APP_CPU_REALTIME_TIME | N | Y | N | N | N |

| メトリック | Linux | HP-UX | Solaris | AIX | Windows |
|---|---|---|---|---|---|
| APP_CPU_REALTIME_UTIL | N | Y | N | N | N |
| APP_CPU_SYS_MODE_TIME | N | Y | N | N | N |
| APP_CPU_SYS_MODE_UTIL | N | Y | N | N | N |
| APP_CPU_TOTAL_TIME | N | Y | N | N | N |
| BYCPU_CPU_SYS_MODE_UTIL | N | Y | N | N | N |
| BYCPU_CPU_TOTAL_UTIL | N | Y | N | N | N |
| BYCPU_CPU_USER_MODE_UTIL | N | Y | N | N | N |
| BYCPU_CSWITCH_RATE | N | Y | N | N | N |
| BYCPU_ID | N | Y | N | N | N |
| BYCPU_INTERRUPT_RATE | N | Y | N | N | N |
| BYCPU_STATE | N | Y | N | N | N |
| PROC_CPU_CSWITCH_TIME | N | Y | N | N | N |
| PROC_CPU_CSWITCH_UTIL | N | Y | N | N | N |
| PROC_CPU_INTERRUPT_TIME | N | Y | N | N | N |
| PROC_CPU_INTERRUPT_UTIL | N | Y | N | N | N |
| PROC_CPU_NICE_TIME | N | Y | N | N | N |
| PROC_CPU_NICE_UTIL | N | Y | N | N | N |
| PROC_CPU_NORMAL_TIME | N | Y | N | N | N |
| PROC_CPU_NORMAL_UTIL | N | Y | N | N | N |
| PROC_CPU_REALTIME_TIME | N | Y | N | N | N |
| PROC_CPU_REALTIME_UTIL | N | Y | N | N | N |
| PROC_CPU_SYSCALL_TIME | N | Y | N | N | N |
| PROC_CPU_SYSCALL_UTIL | N | Y | N | N | N |
| PROC_CPU_SYS_MODE_TIME | N | Y | N | N | N |
| PROC_CPU_SYS_MODE_UTIL | N | Y | N | N | N |
| PROC_CPU_TOTAL_TIME | N | Y | N | N | N |
| PROC_CPU_TOTAL_TIME_CUM | N | Y | N | N | N |
| PROC_CPU_TOTAL_UTIL | N | Y | N | N | N |

| メトリック | Linux | HP-UX | Solaris | AIX | Windows |
|---|---|---|---|---|---|
| PROC_CPU_TOTAL_UTIL_CUM | N | Y | N | N | N |
| PROC_CPU_USER_MODE_TIME | N | Y | N | N | N |
| PROC_CPU_USER_MODE_UTIL | N | Y | N | N | N |

## コアに基づいた正規化によって計算されたメトリックの記録

HP-UX では、Performance Collection Component で、コアに基づいて正規化された CPU 関連のすべてのメトリックを記録するように設定できます。他のプラットフォームでは、Performance Collection Component で、コアに基づく正規化を使用した GLOBAL クラスの CPU 関連のメトリックを計算してから記録するように設定できます。

Performance Collection Component で、CPU 関連のすべてのメトリックにコアに基づく正規化を使用するように設定するには、以下の手順を実行します。

*HP-UX の場合 :*

root アカウントでシステムにログオンします。

ご使用の環境に合わせて parm ファイルを構成します。parm ファイルには ignore_mt フラグを設定しないでください。

> **注 :** HP-UX での parm ファイルの ignore_mt フラグの値は、Performance Collection Component の操作にまったく影響しません。

必要に応じてアラームのルールを定義します。

次のコマンドを実行します。

```
/opt/perf/bin/midaemon -ignore_mt
```

以下のコマンドを実行し、HP Operations エージェントを起動します。

**/opt/OV/bin/opcagt –start**

Performance Collection Component は、コアに基づく正規化を用いた CPU 関連のメトリック (すべてのクラス) の記録を開始します。

HP Operations エージェントを再起動すると、Performance Collection Component はスレッドを基にした正規化を使用してCPU データを再びログに記録し始めるため、前述の手順を使用してもう一度 Performance Collection Component を設定する必要があります。コアに基づく正規化を常に使用するようにエージェントを有効にするには、以下の手順を実行します。

エージェント ノードで、次の場所に移動します。

**/var/opt/perf**

テキスト エディタで次のファイルを開きます。

**vppa.env**

MIPARMS パラメータを ignore_mt に設定します。

ファイルを保存します。

以下のコマンドを実行してエージェントを再起動します。

**/opt/OV/bin/opcagt –start**

*他のプラットフォーム*

root アカウントまたは管理者アカウントでシステムにログオンします。

ご使用の環境に合わせて parm ファイルを構成します。parm ファイルの ignore_mt フラグを true に設定します。

必要に応じてアラームのルールを定義します。

以下のコマンドを使用して、HP Operations エージェントを起動します。

*Windows の場合*

**%ovinstalldir%bin\opcagt -start**

*HP-UX、Linux、Solaris の場合*

**/opt/OV/bin/opcagt -start**

*AIX の場合 :*

**/usr/lpp/OV/bin/opcagt -start**

Performance Collection Component は、コアに基づく正規化を用いた、GLOBAL クラスの CPU 関連のメトリックの記録を開始します。

# データ収集の終了と再開始

scope コレクタとその他の関連プロセスは、継続的に実行される仕様になっています。これらを終了する必要があるのは、次のいずれかの場合だけです。

Performance Collection Component ソフトウェアをバージョンアップするとき

トランザクション構成ファイル、**ttd.conf** 内でトランザクションを追加または削除するとき(詳細は、「トランザクション追跡の紹介」を参照してください)

トランザクション構成ファイル、**ttd.conf** で配布範囲やサービスレベルの目標 (SLO) を変更するとき(詳細は、「トランザクション追跡の紹介」を参照してください)

**parm** ファイルを変更し、その変更内容を反映させるとき。**parm** ファイルに対して行った変更は、scope を開始したときに有効になります。

**utility** プログラムの resize コマンドを使用して、Performance Collection Component ログファイルのサイズを再変更するとき

システムを終了するとき

ハードウェアの追加、または構成の変更を行うとき。変更は、scope を開始したときに有効になります。

## データ収集の終了

**ovpa** および mwa スクリプトの stop オプションを使用すると、scope とその他の Performance Collection Component プロセスの終了時にデータが失われないことが保証されます。

データ収集を手動で終了するには、次のコマンドを入力します。

*Windows の場合:*

*%ovinstalldir%***bin\ovpacmd -stop**

*HP-UX、Linux、Solaris の場合:*

**/opt/perf/bin/ovpa -stop**

*AIX の場合:*

**/usr/lpp/perf/bin/ovpa -stop**

**注: scope** は NFS データを記録しませんが、ローカルファイルシステム上の GlancePlus を介することで NFS データを表示できます。

# データ収集の再開始

Performance Collection Component プロセスが終了しているときや、構成ファイルを変更し、変更内容を反映させるときにデータ収集を再開始するには、次の方法があります。

**coda** を使用している場合、システムがダウンした後に scope とその他の Performance Collection Component のプロセスを開始するには、 **<インストールディレクトリ>/ovpa start** を使用します。ここで、インストールディレクトリは、Performance Collection Component がインストールされたディレクトリです。

**coda** を使用している場合、システムがダウンした後に scope と実行中のその他のプロセスを開始するには **<InstallDir>/ovpa restart** を使用します。ここで、インストールディレクトリは、Performance Collection Component がインストールされたディレクトリです。これにより、現在実行中のプロセスが終了し、再開始します。

scope を再開始すると、Performance Collection Component は、そのプログラムを終了する前に使用していたログファイル (**logglob**、**logappl**、**logproc**、**logdev**、**logtran**、**logls**、**logindx**) を使用し続けます。新しいレコードは既存のファイルの最後に追加されます。新しいファイルセットにデータを収集し、履歴情報を保持しない場合は、再開始する前にすべての scope ログファイル名を変更するか、アーカイブして削除する必要があります。これは、ファイル間でデータを同期させないためです。

**注: ttd.conf** 構成ファイルの SEM_KEY_PATH エントリを使用すると、UNIX プラットフォームの ttd および midaemon プロセスで使用されるセマフォの IPC キーを生成できます。使用されるデフォルト値は、/var/opt/perf/datafiles です。sem id の競合が原因して midaemon または ttd が返答しない場合は、SEM_KEY_PATH の値を変更できます。

# 夏時間

夏時間の間は、該当するタイムゾーン内の環境ではシステムの時刻を 1 時間戻します。この時点で、最後に記録されたレコードのタイムスタンプにシステムの時刻が同期するまで、データ収集が 1 時間にわたって停止します。

夏時間を無効に設定すると、システムの時刻が1時間進みますので、次に記録されるレコードのタイプスタンプが1時間進んでいることになります。この場合は、データ収集が停止しなくても、最後に記録されたレコードの後に1時間のギャップが生じます。

## システムの時刻の手動変更

システムの時刻を手動で戻すと、データ収集が終了し、コマンド (**perfstat** など) が停止します。これらのユーティリティはシステムの時刻を戻した場合に停止します。データの記録を続行し、すべてのコマンドからの応答を取得するには、以下の手順を実行します。

次のコマンドを実行します。

```
ovc -stop coda
```

<データディレクトリ>**datafiles**\ ディレクトリの **coda.*** ファイルをバックアップしてから削除します。

次のコマンドを実行します。

```
ovc -start coda
```

# 効果的なデータ収集管理

パフォーマンス分析の効率は、収集するパフォーマンスデータへのアクセスがどの程度簡略化されているかによって異なります。この項では、データ収集プロセスを簡略化し、より効果的で便利なものにするために行う、ログファイルの管理、データのアーカイブ、システム分析などのアクティビティに対する効果的な方策について説明します。

## ログファイルにより使用されるディスクスペースの制御

Performance Collection Component は、作成したログファイルの自動管理機能を備えています。ユーザーは、この自動プロセスを設定しても、特殊な目的に合わせて手動による代替プロセスを使用してもかまいません。自動ログファイル管理プロセスは次のように機能します。

各ログファイルには、最大サイズが設定されています。Performance Collection Component の初期インストール時にはデフォルトの最大サイズが使用されます。ただし、デフォルト値の設定は変更できます。

各ログファイルがその最大サイズに達すると、scope データコレクタが mainttime で指定された時刻に「ロールバック」を実行します。このロールバックでは、ログファイル内で最も古いものから 25% のデータが削除され、新しいデータを追加するための空き領域が作られます。

自動ログファイルの保守は、scope や DSI の記録プロセスによって収集されるデータと似ていますが、いくつかの相違点があります。DSI ログファイルの保守に関する詳細については、「データソース統合の概要」を参照してください。

### mainttime の設定

通常、`scope` は、毎日指定された時刻にログファイルのロールバックを実行します。したがって、ロールバックをオフピークの時間帯に実行すると、通常のシステム使用率に影響を与えることはありません。ロールバックに関してログファイルを調べる時刻は、`parm` ファイル内の `mainttime` パラメータで設定します。

## ログ ファイルの最 大 サイズの設 定

ログ ファイルの最 大 サイズは、ディスク スペースの使 用 量と、直ちに分 析 可 能な履 歴 データの量とのバランスを考 慮して決 定します。ログ ファイルのサイズを小 さくするとディスク スペースを節 約 できますが、Performance Manager などのツールによってグラフを作 成 できる期 間 が短くなります。次 に、scope ログ ファイルのサイズを変 更 する方 法をいくつか説 明します。

scope は、さまざまなデータ型 を別 々 のログ ファイルに記 録します。このため、各 種 のデータに割 り当てるディスク スペースを決 定 できます。たとえば、グローバル データはかなりコンパクトですが、1 か月 間のデータを振 り返 ってグラフを作 成 する場 合 があります。これにより、傾 向 変 動 分 析 や容 量 計 画を行うための基 本 的 な統 計 データを作 成 できます。

プロセス データは、1 分 ごとに多 数 の対 象 プロセスが発 生 する可 能 性 があるため、グローバル データよりも多くのディスク スペースを消 費します。また、プロセス データの場 合 、時 間 値 がグローバル データの場 合 ほど高くありません。どのようなプロセスが今 日 と昨 日 実 行 されたかについて詳 細を把 握 することが極 めて重 要 になります。場 合 によっては、先 週 実 行 されたプロセスを把 握 する必 要 もあります。ただし、先 月 実 行 されたプロセスを正 確 に把 握 していても役 立 つことはほとんどありません。

一 般 に、ユーザーがオンラインで保 持 するデータは次 のようなものです。

3 か月 分 のグローバル データ (傾 向 変 動 分 析 のためのデータ)

1 か月 分 のプロセス データ (トラブルシューティングのためのデータ)

3 か月 分 のアプリケーション データ (傾 向 変 動 分 析 および負 荷 配 分 のためのデータ)

2 か月 分 のデバイス データ (ディスク負 荷 配 分 のためのデータ)

**parm** ファイルを編 集 して、各 ログ ファイルの **size** パラメータを設 定 できます。サイズはメガバイト単 位 で指 定 します。たとえば、次 のようになります。

```
SIZE GLOBAL=10.0 PROCESS=30.0 APPLICATION=20.0 DEVICE=5.0
```

特 定 の日 数 のデータを保 持 するために必 要 なメガバイト数 は、データ型 、システム構 成 、およびシステムの活 動 状 況 によって異 なります。システムのログ ファイルの大 きさを決 定 する最 良 の方 法 は、1 週 間 ほどデータを収 集 した後 、**utility** プログラムの resize コマンドを使 用 してログ ファイルのサイズを変 更 する方 法 です。resize コマンドは、ログ ファイルを走 査 して、1 日 に収 集 されるデータ量 を確 認 します。次 に、日 数 からメガバイト数 を自 動 的 に算 出 します。この関 数 は **parm** ファイルも更 新 します。

## サイズ変 更 プロセスの管 理

ログ ファイルの自 動 保 守 を設 定 すると、他 の設 定 作 業 は不 要 になります。ログ ファイルが、設 定 されている最 大 サイズに達 すると、ログ ファイルのサイズが scope によって自 動 的 に変 更 されます。

scope は、**parm** ファイルの mainttime で指 定 されている時 刻 に、ログ ファイルをロール バックします。**parm** ファイルを編 集 して scope を再 開 始 しても、ログ ファイルは mainttime の時 刻 になるまで新しいサイズに変 更 されません。mainttime で指 定 した時 刻 に scope を実 行 していることが重 要 です。実 行 していないと、ログ ファイルはロール バックされません。

保 守 時 刻 以 外 の時 間 にログ ファイルが最 大 サイズの設 定 値 を超 えても、ロール バックは行 われません。

ログ ファイルは、丸 1 日 分 のデータを保 持 できないサイズに変 更 されることはありません。つまり、ログ ファイルは、少 なくとも 1 日 分 のデータを保 持 するサイズまで増 大 してからロール バックされます。通

常、このことが問題になることはありませんが、大量のプロセス データやアプリケーション データを収集するように **parm** ファイルのパラメータを設定している場合、またはそのサイズを小さく設定している場合は、ログ ファイルがロール バックされる前にログ ファイルのサイズが最大サイズの設定値を大幅に超えてしまう可能性もあります。

scope は、ログ ファイルが存在しているファイル システムの空きディスク スペースを、グローバル データ収集用に **parm** ファイルに指定されている間隔で定期的に確認します。空きスペースが 1 メガバイト未満になると、scope は、空きスペースをそれ以上使用しないように次の処理を行います。

定期的なログ ファイル保守時刻を待たず、直ちにログ ファイルの保守を実行します。いずれかのログ ファイルがその最大サイズを超えた場合 (および、そのログ ファイルに 1 日分以上のデータが保持されている場合)、そのログ ファイルはロール バックされます。

ログ ファイルの保守が終わってもディスクの空きスペースが 1 メガバイト以下の場合、scope は、適切なエラー メッセージをその status.scope ファイルに書き込み、データの収集を終了します。

## データのアーカイブ

自動ログ ファイル管理は、分析に使用できる最新のログ ファイル データを保持します。生ログ ファイルからのデータをアーカイブします。プロセス データおよびグローバル データは、parm ファイルで指定した間隔で定期的に記録されます。詳細情報は、「データ記録間隔の設定」を参照してください。新しいデータのスペースを確保するために、ログ ファイルが最大サイズに達すると最も古いデータから削除されます。ログ ファイル データの保存期間を長くする場合、データのアーカイブ処理を行います。どのような処理を行うかは、ユーザーの必要性によって異なります。次に、いくつかの方法を示します。

生ログ ファイルのサイズを非常に大きくして、残りの処理は自動ログ ファイル保守に任せます。これは、最も簡単なアーカイブ方法ですが、数か月後には大量のディスク スペースを消費している可能性があります。

生ログ ファイルから削除される前に、データを抽出アーカイブ ファイルに抽出します。このアーカイブ ファイルは、将来必要になるときに備えて、テープなどの長期記憶装置にコピーしてください。

生ログ ファイルのサブセットだけを抽出アーカイブ ファイルに抽出します。たとえば、容量が大きくて時間値が低いためにプロセス データがアーカイブされない場合があります。

上記の方法を組み合わせて使用する方法もあります。

データのアーカイブは次の方法で行うことをお勧めします。

オンラインで保持する詳細データの量が収まるように生ログ ファイルのサイズを決定します。

週に 1 度、オフライン記憶装置に移すファイルに詳細な生データをコピーします。

## アーカイブ処理の管理

前の項の説明に従って、生ログ ファイルのサイズを変更してください。最低 2 週間分のデータを保持できるログ ファイル サイズを選択します (アーカイブ処理を週に 1 度だけ行うと仮定します)。

週に 1 度、**extract** プログラムを実行するプロセスをスケジュールします。以下に、毎週処理を実行するスクリプト ファイルのサンプルを示します。

```
#extract -gapdt -xm
```

その月の各週のデータが、前の週のデータに追加されます。新しい月になると、**extract** は新しいアーカイブ ログ ファイルを作成し、その週のデータを該当する月間アーカイブ ログ ファイルに分割します。ロ

グ ファイルには **rxmo** という名 前 が付 けられ、その後 に年 を表 す 4 桁 の数 字 と、月 を表 す 2 桁 の数 字 が続 きます。(たとえば、1999 年 12 月 に抽 出 したデータは、**rxmo199912** ファイルに記 録 されま す)。

月 が変 わると前 の月 のログ ファイルを終 了 し、新 しいログ ファイルを開 始 します。したがって、ログ ファ イル **rxmo** が複 数 存 在 する場 合 、最 新 でないファイルは、必 要 になるまでオフラインの記 憶 装 置 にコ ピーしておく方 が安 全 です。アーカイブ データにアクセスする必 要 があるときは、対 象 となるアーカイブ ファイルを復 元 して、**extract** プログラムまたは **utility** プログラムを使 用 してアクセスしてください。

システム構 成 や活 動 レベルによっては、1 か月 のディスク スペースの累 積 使 用 量 が大 きくなることもあ ります。このような場 合 は、上 記 の例 に示 されている -xm コマンド の代 わりに -xw コマンド を使 用 する ことで、詳 細 なアーカイブ ファイルを複 数 の小 さなファイルに細 分 化 できます。

また、詳 細 なプロセス データをアーカイブしないようにも選 択 できます。

前 記 の例 で説 明 した詳 細 の抽 出 では、収 集 したパフォーマンス データがすべて保 存 されます。ある 状 況 を詳 しく調 べる必 要 がある場 合 、これらのファイルをディスクに復 元 して分 析 できます。

**ヒント :** extract プログラムを使 用 すると、転 送 や分 析 を容 易 にするために、複 数 の抽 出 ファイル の結 合 データやデータのサブセットを作 成 できます。たとえば、複 数 年 の傾 向 変 動 分 析 を行 うた めに、複 数 の年 間 抽 出 ファイルを結 合 できます

**注 :** HP Operations エージェントで作 成 されたログ ファイルを、古 いバージョンの HP Performance Agent を使 用 しているシステムに移 動 することはできません。

# HP Operations エージェント の操作

データ収集機構を設定した後、そのエージェントを HPOM と共に使用する場合、HPOM のポリシーをノードに配布することにより運用監視コンポーネントの異なるコンポーネントを使用できます。たとえば、Measurement Threshold ポリシーを配布すると、監視エージェント コンポーネントは動作を開始します。ほとんどの監視詳細は HPOM ポリシーに指定できますが、そのノード上でさらに設定を行う必要があるコンポーネントもあります。

## 監視エージェントの設定

さまざまなソースを監視する監視エージェントを起動/設定できます。ノードに Measurement Threshold ポリシーを配布すると、監視エージェント コンポーネントは動作を開始します。エージェントは、ポリシーに指定されている内容に基づいて、以下のソースのオブジェクトの監視を開始します。

- **外部 :** エージェントに数値を送信できる外部プログラム。
- **組み込みパフォーマンス コンポーネント :** エージェントのデータ ストアに存在するデータ。
- **MIB:** 管理情報ベース (MIB) のエントリ。
- **リアルタイム パフォーマンス管理 :** Windows のパフォーマンス ログおよびアラート。
- **プログラム:** HPOM によって起動され、エージェントに数値を送信する外部プログラム。
- **WMI:** WMI データベース。

HPOM ポリシーを使用して上記ソースのオブジェクトを監視する方法については、以下のトピックを参照してください。

- *HPOM for Windows:HPOM for Windows* のオンライン ヘルプの「*Event Policy Editors*」
- *HPOM on UNIX/Linux:『HPOM for UNIX 9.10 Concepts Guide』*の「*Implementing Message Policies* 」

## MIB オブジェクトを監視するエージェントの構成

ノードに Measurement Threshold ポリシー (ソース タイプを MIB に設定 ) を配布すると、監視エージェントは、public というコミュニティ文字列でアクセスできる MIB オブジェクトの照会を開始します。デフォルト以外のコミュニティ文字列を使用するように監視エージェントを構成する手順は以下のとおりです。

1. ルート権限または管理権限でノードにログオンします。
2. コマンド プロンプト (シェル) に移動します。
3. 以下のディレクトリに移動します。

   *Windows の場合 :*

   *%ovinstalldir%***bin**

   *HP-UX、Solaris、Linux の場合 :*

   **`/opt/OV/bin`**

   *AIX の場合 :*

```
/usr/lpp/OV/bin
```

4. 次のコマンドを実行します。

- デフォルト以外のコミュニティ文字列を使用するには
  **ovconfchg -ns eaagt SNMP_COMMUNITY** *<コミュニティ文字列>*

  この *<コミュニティ文字列>* には、デフォルト以外のコミュニティ文字列を指定します。

- 別のコミュニティ文字列を使用するには
  **ovconfchg -ns eaagt SNMP_COMMUNITY_LIST** *<コミュニティ文字列のリスト>*

  この *<コミュニティ文字列のリスト>* には、使用するコミュニティ文字列のカンマ区切りのリストを指定します。HP Operations エージェントは、上記コマンドに指定された順にコミュニティ文字列のリストを処理します。

  たとえば、次のようになります。

  ```
  ovconfchg -ns eaagt SNMP_COMMUNITY_LIST "C1,C2,C3"
  ```

  HP Operations エージェントは、ノードとの SNMP セッションを確立し、まず、コミュニティ文字列 C1 を使用して OID に対して SNMP Get 処理を試みます。この操作が失敗すると、HP Operations エージェントはコミュニティ文字列 C2 を使って同じ操作を実行し、成功するまで順にこれを繰り返します。

  > **注 :** SNMP_COMMUNITY_LIST に指定されているすべてのコミュニティ文字列を使っても HP Operations エージェントが失敗する場合は、SNMP_COMMUNITY に指定されているコミュニティ文字列が使用されます。指定されているすべてのコミュニティ文字列を使ってもエージェントがデータを取得できない場合は、デフォルトのコミュニティ文字列である public が使用されます。

# 監視対象オブジェクトの永続化

HP Operations エージェントでは、中断や障害のイベントに備えて、監視対象オブジェクトとセクション変数の値を定期的に保存します。

ただし、監視対象オブジェクトとセッション変数の値を保存しないようにエージェントを設定することもできます。

ただし、監視対象オブジェクトとセッション変数の値を保存しないようにエージェントを設定することもできます。 監視対象オブジェクトとセッション変数の値を保存すると、中断や障害の発生時に保存した値を利用できます。

OPC_MON_SAVE_STATE 変数を使うと、監視対象オブジェクトとセッション変数の値を保存するようにエージェントを設定できます。

監視対象オブジェクトとセッション変数の値を定期的に保存するようにエージェントを設定するには、以下の手順に従います。

監視した値を保存しない場合は、以下の手順に従います。

## 監視対象の値を保存しないようにエージェントを設定するには

1. root または管理者としてノードにログオンします。
2. 次のコマンドを実行します。

```
ovconfchg -ns eaagt -set OPC_MON_SAVE_STATE TRUE

ovconfchg -ns eaagt -set OPC_MON_SAVE_STATE FALSE
```

エージェントは、監視対象オブジェクトとセッション変数の値の保存を停止開始します。

HP Operations エージェント 11.10 をインストールすると、OPC_MON_SAVE_STATE 変数に次のような影響を与えます。

- HP Operations エージェント 11.10 をインストールする前に変数を任意の値に設定しなかった場合、OPC_MON_SAVE_STATE 変数は値を FALSE と見なします。
- HP Operations エージェント 11.10 をインストールする前に ovconfchg コマンドを使用して値 (TRUE または FALSE) を設定した場合、インストール プロセスの終了後も設定した値は有効のままになります。

たとえば、バージョン 11.10 をインストールする前に **ovconfchg -ns eaagt -set OPC_MON_SAVE_STATE TRUE** コマンドを使用した場合、HP Operations エージェント 11.10 のインストール後にも同じ値 (TRUE) が維持されます。

# イベント インターセプタの構成

イベント インターセプタは、デフォルトではリモート管理ステーションまたは SNMP 対応デバイスから出力される SNMP トラップを収集し、構成に基づいて適切なイベントを生成できます。イベント インターセプタのデフォルト動作は、以下のプロパティの構成によって変更できます。

- SNMP_TRAP_PORT: デフォルトのポートは **162** です。この値は、HP Operations エージェントノードの任意のポート番号に変更できます。
- SNMP_TRAP_FORWARD_DEST_LIST: このプロパティを使用することで、使用できるすべての SNMP トラップの転送先となるリモート管理ステーションのアドレスを設定できます。カンマで区切ることで、複数のシステム名 (およびポートの詳細) を指定できます。
- SNMP_TRAP_FORWARD_ENABLE: デフォルトでは、このプロパティは **FALSE** に設定されています。このプロパティを **TRUE** に設定した場合、イベント インターセプタは、HP Operations エージェントノードで使用できる SNMP トラップをリモート マシンまたは管理ステーションに転送できるようになります。
- SNMP_TRAP_FORWARD_COMMUNITY: このプロパティを使用することで、受信トラップのソース マシンと、SNMP トラップの転送先となるターゲット マシンのコミュニティ文字列を指定できます。ソース マシンとターゲット マシンのコミュニティ文字列は、一致している必要があります。
- SNMP_TRAP_FORWARD_FILTER: このプロパティを使用することで、使用できる SNMP トラップを OID でフィルタ処理し、選択されたトラップのみをリモート マシンに転送できます。このフィルタ メカニズムでは、ワイルドカード文字 (*) が有効です。たとえば、このプロパティを **1.2.3.*.*.*** に設定した場合、イベント インターセプタは OID が 1.2.3 から始まるすべての SNMP トラップを転送します。イベント インターセプタによるトラップの転送を有効にした場合、デフォルトでは、使用できるすべてのトラップが転送されます。

> **注:** ソース マシンとターゲット マシンのコミュニティ文字列が一致しない場合、トラップ転送機

能は失敗します。

イベント インターセプタのデフォルト動作を変更する手順は以下のとおりです。

1. 適切な権限でノードにログオンします。
2. コマンド プロンプトで以下のコマンドを実行します。

- ポート番号を変更するには、以下のコマンドを実行します。

**ovconfchg -ns eaagt set SNMP_TRAP_PORT** *<ポート番号>*

*<ポート番号>* には整数値を指定する必要があります。指定する *<ポート番号>* が使用可能であることを確認してください。

- イベント インターセプタが SNMP トラップをリモート マシンに転送できるようにするには、以下のコマンドを実行します。

**ovconfchg -ns eaagt set SNMP_TRAP_FORWARD_ENABLE TRUE**

- イベント インターセプタが SNMP トラップをリモート マシンに転送できるようにする場合は、以下のコマンドを実行してターゲット マシンの詳細を指定します。

**ovconfchg -ns eaagt set SNMP_TRAP_FORWARD_DEST_LIST "***<マシン名>:<ポート>***"**

*<マシン名>* には SNMP トラップの転送先となるマシンの完全修飾ドメイン名、*<ポート>* にはそのマシンの HTTPS ポートを指定します。複数のターゲット マシンを指定するには、マシンの詳細をカンマで区切ります。

- ノードで使用できる SNMP トラップのうち、選択したトラップのみをリモート マシンに転送するには、以下のコマンドを実行します。

**ovconfchg -ns eaagt set SNMP_TRAP_FORWARD_FILTER "***<OID フィルタ>***"**

*<OID フィルタ>* には、OID とワイルドカード文字を指定します。イベント インターセプタは、使用できるトラップから指定の OID (およびワイルドカード文字) と一致するトラップをフィルタリングし、これらのトラップのみをターゲット マシンに転送します。

イベント インターセプタを構成し、外部ソースから発信された SNMP トラップを待ち受けるためにポート 162 にバインドする異なるメカニズムを使用することができます。SNMP_SESSION_MODE 変数を使用すると、この構成が可能になります。

- Net-SNMP API を使用してポート 162 にバインドし SNMP トラップを待ち受けるようイベント インターセプタを構成するには、以下のコマンドを実行します。
- **`ovconfchg -ns eaagt set SNMP_SESSION_MODE NETSNMP`**
- *Windows ノードのみ。* イベント インターセプタを構成し、SNMP トラップの待ち受けに Microsoft トラップ サービスを使用するには、以下のコマンドを実行します。
- **`ovconfchg -ns eaagt set SNMP_SESSION_MODE WIN_SNMP`**
- *UNIX/Linux ノードのみ。* OVSNMP API を使用してポート 162 にバインドし SNMP トラップを待ち受けるようイベント インターセプタを構成するには、以下のコマンドを実行します。
- **`ovconfchg -ns eaagt set SNMP_SESSION_MODE NO_TRAPD`**

- OVSNMP API を使用してポート 162 にバインドし SNMP トラップを待ち受けるようにイベント インターセプタを構成するには、以下のコマンドを実行します。
- **`ovconfchg -ns eaagt set SNMP_SESSION_MODE NNM_LIBS`**
- SNMP トラップを待ち受けるのに、NNM (7.5x) の PMD デーモンに登録を試みるようイベント インターセプタを構成するには、以下のコマンドを実行します。
- **`ovconfchg -ns eaagt set SNMP_SESSION_MODE TRY_BOTH`**
- イベント インターセプタはまず NNM の PMD デーモンへの登録を試みますが、失敗した場合は、OVSNMP API を使用してポート 162 にバインドします。
- *On UNIX/Linux ノードのみ。* SNMP トラップを待ち受けるのに、NNM (7.5x) の PMD デーモンに登録するようイベント インターセプタを構成するには、以下のコマンドを実行します。
- **`ovconfchg -ns eaagt set SNMP_SESSION_MODE NNM_PMD`**

# RTMA コンポーネントの構成

リアルタイム メトリック アクセス (RTMA) コンポーネントにより、ローカルまたはリモートで、システム パフォーマンス メトリックにリアルタイムでアクセス可能となります。HP Operations エージェントをインストールすると、RTMA コンポーネントの一部である perfd プロセスは、デフォルトの構成でノードで動作を開始します。perfd プロセスの設定は **perfd.ini** ファイルで変更できます。このファイルは、ノードの以下のディレクトリにあります。

Windows の場合 :*%ovdatadir%*

UNIX (Linux) の場合 :**/var/opt/perf**

| パラメータ | 説明 | デフォルト値 |
|---|---|---|
| interval | データ収集の頻度 (単位は秒)。この値は、60 の倍数または約数である必要があります。 | 10 |
| port | perfd が使用するポート。 | 5227 |
| depth | perfd キャッシュでグローバル メトリック値を維持する時間。このデータは、データの要約に使用されます。 | 30 |
| maxrps | perfd が 1 秒間に受け付けるセッション要求の最大数。要求の数がこの制限を超えると、perfd は 1 秒間停止し、このイベントの詳細をログ ファイルに記録します。このログ ファイル (status-perfd.<ポート>) は、ノードの以下のディレクトリにあります。<br>*Windows の場合 : %ovdatadir%* | 20 |

| パラメータ | 説明 | デフォルト値 |
|---|---|---|
| | *UNIX (Linux) の場合 :***/var/opt/perf** | |
| maxtpc | perfd が受け付ける、クライアント システムごとのセッションの最大数。使用可能セッションの数がこの制限に達した状態で、追加の要求を受信した場合、perfd はその要求を拒否します。 | 30 |
| maxcps | perfd が一度に受け付ける同時セッション要求の最大数。要求の数がこの制限を超過すると、サーバーは 3 秒間停止してからセッションを確立します。 | 2 |
| lightweight | このパラメータを true に設定すると、perfd はプロセス、アプリケーション、NFS 操作、論理システム、ARM のデータ収集を停止します。また、HP-UX 上の HBA および LVM データも収集されなくなります。 | false |
| localonly | このプロパティを true に設定すると、perfd への接続をローカル マシンのみに限定できます。<br><br>true に設定した場合、perfd は、ループバック インターフェイスを経由したホスト システム (ローカル ホスト) からの要求を除く、すべての接続要求を拒否します。拒否された接続要求の詳細は、ステータス ファイルに記録されます。 | false |
| ipv4 | このオプションを true に設定すると、perfd は IPv4 接続のみを受け付けるようになります。perfd が IPv6 ソケットを作成できない場合、デフォルトでは、IPv4 のみのソケットに自動的に切り替わります。このオプションは、すべての IPv6 通信を明示的に無効にする場合に使用されます。 | false |

デフォルト設定を変更する手順は以下のとおりです。

## デフォルト設定を変更するには

テキスト エディタを使用して、ノードの **perfd.ini** ファイルを開きます。

設定を変更します。

ファイルを保 存します。

HP Operations エージェントを再 起 動して設 定 変 更を有 効にします。

# アクセスの制 限

RTMA コンポーネントを使 用して、ローカル システムのリアルタイム パフォーマンス データに他 のシステムがアクセスしないように HP Operations エージェントを設 定 できます。

cpsh、padv、mpadv ユーティリティを使うと、エージェント ノード のリアルタイム パフォーマンス データにリモート でアクセスできます。

cpsh ユーティリティを使 用して、perfd プロセスが実 行 されている任 意 のリモート システムにあるリアルタイム パフォーマンス データを表 示 できます。

padv または mpadv ユーティリティを使 用して、perfd プロセスが実 行 されている任 意 のリモート システムで adviser を実 行 できます。

これらのユーティリティはリアルタイム パフォーマンス データにアクセスするために perfd プロセスに依 存します。

他 のシステムが perfd プロセスを起 動 することによってローカル システムのリアルタイム パフォーマンス データにアクセスしないようにするには、以 下 の手 順 に従 います。

| | |
|---|---|
| | この手 順 では、HP Performance Manager の Diagnostic View から、システムのリアルタイム パフォーマンス データへのアクセスを禁 止 するようには設 定 されません。 |

管 理 者 または root としてシステムにログオンします。

以 下 のディレクトリに移 動します。

*Windows の場 合*

```
%ovdatadir%conf\perf
```

*UNIX/Linux の場 合*

```
/var/opt/OV/conf/perf
```

この場 所 に空 のファイルを作 成して、そのファイルを **authip** として保 存します。

環 境 にある (HP Operations エージェントが利 用 可 能 な) 他 のすべてのシステムからは、このシステムのリアルタイム パフォーマンス データにアクセスできなくなりました。

このシステムのリアルタイム パフォーマンス データに別 のシステムからアクセスできるようにするには、テキスト エディタで **authip** ファイルを開 いて、システムの FQDN または IP アドレスを追 加し、ファイルを保 存します。複 数 の FQDN または IP アドレスを指 定 できます (エントリは改 行 で区 切 る必 要 があります)。

**authip** ファイルで指 定 したシステムは、このシステムのリアルタイム パフォーマンス データにアクセスできるようになりました。

すべてのシステムからアクセスできるようにするには、**authip** ファイルを削 除します。

# エージェント ユーザーの設定

インストール後、HP Operations エージェントは Windows ノードではローカル システムのアカウントで、UNIX/Linux ノードでは root アカウントで稼働し始めます。ただし、root やローカル システム ユーザーよりも少ない権限を持つデフォルト以外のユーザーで実行するように HP Operations エージェントを設定できます。

運用監視コンポーネント は root 以外のユーザーやローカル システム ユーザー以外のユーザーだけで実行し、残りのコンポーネントは、デフォルトの rootまたはローカル システム ユーザーで実行することもできます。

使用するユーザー アカウントに応じて、次のエージェント操作モードを設定できます。

- **権限のないモード :** HP Operations エージェントのすべてのコンポーネントは、root やローカル システム ユーザーよりも少ない権限を持つデフォルト以外のユーザー アカウントで実行されます。

  **注 :** HP-UX では権限のないモードで HP Operations エージェント を実行することはできません。

- **root モード :** HP Operations エージェントのすべてのコンポーネントは、デフォルト ユーザー (root またはローカル システム)で実行されます。これは、エージェントの操作モードのデフォルトです。
- **混合モード :** 運用監視コンポーネント だけが、root またはローカル システム以外のアカウントで実行され、HP Operations エージェントの他のすべてのコンポーネントは root またはローカル システムで実行されます。

  **注 :** 混合モードで実行する場合、root または権限のあるユーザーがエージェント プロセスを開始することを推奨します。

HP Performance Manager がインストールされているシステムで、デフォルト以外のユーザーで実行されるようにエージェントが設定されている場合、HP Performance Manager の `OvTomCatB` サービスは、デフォルト以外のエージェント ユーザーで実行が開始されます。

さらに、HPOM 管理対象環境では、エージェントを実行するように設定されているユーザーとは異なるユーザーが自動コマンドまたはオペレータによるコマンドを実行するようにエージェントを設定することもできます。

## デフォルトではないユーザーを使用するための要件

デフォルトではないエージェント ユーザーを使用するには、以下の要件を満たす必要があります。

- **Windows のみの要件 :**
  - ユーザーには `HKEY_LOCAL_MACHINE/Software/Hewlett-Packard/OpenView` レジストリキーのフルコントロールが必要です。
  - ユーザーには `HKEY_LOCAL_MACHINE/Software/Microsoft/WindowsNT/CurrentVersion/Perflib` レジストリキーへの読み取りアクセス権が必要です。

- ユーザーには次の権限が必要です。
  - サービスとしてログオン
  - 監査およびセキュリティログの管理
  - システムのシャットダウン
  - プログラムのデバッグ
  - オペレーティングシステムの一部として動作
  - プロセスのメモリ容量の調節 (Windows の一部のバージョンでは、クォータの増加とも呼ばれる)
  - プロセスレベルトークンの置き換え
- ポリシーを使用してログファイルを監視できるようにする場合、エージェント ユーザーにはログファイルの読み取り権限が必要です。
- 自動コマンド、オペレータ起動コマンド、ツール、予定タスクを使用してプログラムを開始できるようにする場合、エージェント ユーザーにはプログラムを開始する権限が必要です。
- 別のユーザーでエージェントを実行する場合、Smart Plug-in の一部では設定やユーザー権限を追加する必要がある場合があります。詳細については、それぞれの Smart Plug-in のドキュメントを参照してください。

## インストール中のエージェント ユーザーの設定

インストール時に、システムでデフォルト以外のユーザー (root またはローカルシステム以外) で HP Operations エージェントを実行するように設定できます。このように設定するには、手動でのインストールに使用するプロファイルファイルを使うか、HPOM によるリモート インストールに使用するデフォルト ファイルを使ってエージェントをインストールする必要があります。インストール時にこのように設定できない場合は、インストール後の設定手順を実行して、デフォルトのエージェント ユーザーを変更します (「インストール後のエージェント ユーザーの設定」(65ページ) を参照)。

> **注:** HPOM コンソールから Windows ノードにエージェントをリモートでインストールする場合、この手順は使用できません。HPOM コンソールから Windows ノードにエージェントをインストールしているときに、*非アクティブ*モードでエージェントをインストールし、インストール後の設定手順を使用して、デフォルト以外のユーザーでエージェントが実行されるように設定します。詳細については、『*HP Operations エージェント インストール ガイド*』を参照してください。

デフォルト以外のユーザーで実行されるようにインストール中にエージェントを設定するには、以下の手順に従います。

1. ユーザーがシステムで作成されていることおよびとユーザーがすべての要件を満たしていることを確認します。
2. HP Operations エージェントを手動でノードにインストールする場合は、**プロファイル** ファイルを作成します。
   a. エージェントをインストールするシステムで新しいファイルを作成して、そのファイルをテキスト エディタで開きます。
   b. 次の文を指定して、エージェントの操作モードを指定します。

root 以外またはローカル システム 以外 のアカウントでエージェントを実行するには、次のように入力します。

**set eaagt:MODE= NPU**

root 以外またはローカル システム以外 のアカウントで 運用監視コンポーネント のみを実行するには、次のように入力します。

**set eaagt:MODE= MIXED**

c. 次の文を入力します。

**set eaagt:OPC_RPC_ONLY=TRUE**

ここで、*<port_number>* は HP Operations エージェントで SNMP トラップを受け取るポートです。これは 1024 より大きい番号である必要があります。

d. 混合モード (手順 b で `set eaagt:MODE=MIXED` と入力した場合)、次の文を入力します。

**set eaagt:NPU_TASK_SET=EVENT_ACTION**

e. *Windows のみ。* 次の文を入力します。

**set eaagt:OPC_PROC_ALWAYS_INTERACTIVE=NEVER**

f. 次の文を入力します。

> **注 :** これは、UNIX/Linux ノードに必須の手順です。Windows ノードではこの手順を省略できますが、Windows ノードでもこれらの手順を設定することをお勧めします。

**set bbc.cb:SERVER_PORT=***<Comm_Port>*

**set eaagt:SNMP_TRAP_PORT=***<SNMP_Port>*

> **注 :** エージェントのデフォルトの通信ポートは 383 であり、UNIX/Linux の root 以外のユーザーは 1024 未満のポートにアクセスできないため、この手順を実行して、デフォルト以外の通信ポートをエージェントに割り当てる必要があります。

上記コマンドの説明 :

*<Comm_Port>* は選択した通信ポート番号です。

*<SNMP_Port>* は HP Operations エージェントで SNMP トラップを受け取るポートです。

これらのポートは 1024 より大きい番号である必要があります。UNIX/Linux の root 以外のユーザーは 1024 未満のポートにアクセスできないためです。

g. 次の文を入力します。

**set ctrl.sudo:OV_SUDO_USER=***<User_Name>*

**set ctrl.sudo:OV_SUDO_GROUP=***<User_Group>*

ここで、*<User_Name>* はデフォルト以外のユーザーの名前であり、*<User_Group>* はデフォルト以外のユーザーが属するグループです。

HPOM コンソールからエージェントをリモートでインストールする場合、インストールのデフォルトを次のように設定します。

> **注:** Windows ノードにエージェントをインストールする場合、この手順は使用できません。Windows ノードでは、エージェントをノードに手動でインストールするか、インストール後の設定を実行して、エージェント ユーザーを設定してください。

a. 以下のディレクトリに移動します。

   *HPOM for Windows の場合*

   *<share_dir>*`\conf\PMAD`

   *UNIX/Linux の HPOM の場合*

   `/etc/opt/OV/share/conf/OpC/mgmt_sv`

b. *HPOM for Windows の場合* `agent_install_defaults.cfg.sample` ファイルを `agent_install_defaults.cfg` として保存します。

c. *UNIX/Linux の HPOM の場合* `bbc_inst_defaults.sample` ファイルを `bbc_inst_defaults` として保存します。

d. テキスト エディタでファイルを開きます。

e. 次の内容を追加します。

```
[eaagt]
<node_details> : MODE=<MODE>
<node_details> : OPC_RPC_ONLY=TRUE
<node_details>: NPU_TASK_SET=EVENT_ACTION
<node_details>: SNMP_TRAP_PORT=<SNMP_Port>
[ctrl.sudo]
<node_details> : OV_SUDO_USER=<User_Name>
<node_details> : OV_SUDO_GROUP=<User_Group>
[bbc.cb]
<node_details>: SERVER_PORT=<Comm_Port>
```

   ここで、

   *<node_details>* は1 つまたは複数のノード名または IP アドレスに一致するパターンです。標準の HPOM パターン構文を使用します。たとえば、次のようになります。

   - node1.example.com は、node1.example.com という文字列が含まれる名前を持つ任意のノードに一致します
   - example.com$ は、example.com で終わる名前を持つ任意のノードに一致します

- ^192.168.<<#> -lt 10> は、IP アドレスが 192.168.0.0 から192.168.9.255 の範囲にある任意のノードに一致します

<*MODE*> はエージェントの操作モード (NPU または混合) です

<*User_Name*> ははデフォルト以外のユーザーの名前です

<*User_Group*> はデフォルト以外のユーザーが属するグループです。

<*Comm_Port*> は選択した通信ポート番号です。これは 1024 より大きい番号である必要があります。

<*SNMP_Port*> は HP Operations エージェントで SNMP トラップを受け取るポートです。これは 1024 より大きい番号である必要があります。

f. ファイルを保存します。

3. エージェントをインストールします。エージェントのインストールの詳細については、『*HP Operations エージェント インストール ガイド*』を参照してください。

# インストール後のエージェント ユーザーの設定

HP Operations エージェントのインストール時に、デフォルトではないユーザーを使用するように設定できない場合、インストール後に `oainstall` スクリプト (エージェント ノード上で利用可能) の `-configure` オプションまたは `ovswitchuser` コマンドを使用して、この設定を完了できます。

## Windows でのデフォルト ユーザーの変更

インストール時に、デフォルト以外のユーザーでエージェントを実行するように設定できない場合は (「インストール中のエージェント ユーザーの設定」(62ページ) を参照)、*非アクティブ*モードでインストールすることをお勧めします。詳細については、『*HP Operations エージェント インストール ガイド*』を参照してください。

デフォルト以外のユーザーを使用してエージェントを設定するには、次の方法を使用できます。

- プロファイル ファイルの使用
- ovswitchuser コマンドの使用

### プロファイル ファイルの使用

プロファイル ファイルを使ってデフォルト エージェント ユーザーを変更するには、次の手順を実行します。

1. エージェントをインストールするシステムで新しいファイルを作成して、そのファイルをテキスト エディタで開きます。
2. 次の文を指定して、エージェントの操作モードを指定します。

   ローカル システム以外のアカウントでエージェントを実行するには、次のように入力します。

   **set eaagt:MODE= NPU**

   ローカル システム以外のアカウントで運用監視コンポーネント のみを実行するには、次のように入力します。

**set eaagt:MODE= MIXED**

3. 権限のないモードを選択した場合 (前の手順で `set eaagt:MODE=NPU` と入力した場合)、次の文を入力します。

   **set eaagt:OPC_RPC_ONLY=TRUE**

4. 混合モード (手順 2 で `set eaagt:MODE=MIXED` と入力した場合)、次の文を入力します。

   **set eaagt:NPU_TASK_SET=EVENT_ACTION**

5. 次の文を入力します。

   **set eaagt:OPC_PROC_ALWAYS_INTERACTIVE=NEVER**

6. 次の文を入力します。

   **set ctrl.sudo:OV_SUDO_USER=**<*User_Name*>

   **set ctrl.sudo:OV_SUDO_GROUP=**<*Group_Name*>

   ここで、<*User_Name*> はデフォルト以外のユーザーの名前であり、<*Group_Name*> はデフォルト以外のユーザーが属するグループです。

7. 次の文を入力します。

   > **注:** この手順は UNIX/Linux ノードに必要であり、Windows ノードでは必須ではありません。ただし、Windows ノードでもこの手順を完了することをお勧めします。

   **set eaagt:SNMP_TRAP_PORT=**<*SNMP_port*>

   **set bbc.cb:SERVER_PORT=**<*Comm_port*>

   上記コマンドの説明:

   <*Comm_Port*> は選択した通信ポート番号です。

   <*SNMP_Port*> は HP Operations エージェント で SNMP トラップを受け取るポートです。

   これらのポートは 1024 より大きい番号である必要があります。

8. システムのローカルディレクトリにファイルを保存します。

9. プロファイルファイルで指定したユーザーで実行されるようにエージェントを再設定します。
   a. ノードの以下の場所に移動します。

      Windows 64 ビットノードの場合 `%ovinstalldir%bin\win64\OpC\install`

      他のすべての Windows ノードの場合 : `%ovinstalldir%bin\OpC\install`

   b. 次のコマンドを実行します。

      **cscript oainstall.vbs -a -configure -agent_profile** <*path*>\<*profile_file*> **-npu_password** <*Password*>

      ここで、<*profile_file*> はプロファイルファイルの名前であり、<*path*> はプロファイルファイルへの完全パスであり、<*Password*> はデフォルト以外のユーザーのパスワードです。

## もう一方の方法: ovswitchuser コマンドの使用

プロファイルファイルを使用しない場合は、以下の手順に従います。

**注:** 設定手順としては、プロファイル ファイルでの設定の方をお勧めします。

1. エージェントを停止します。

   *Windows 64 ビット ノードの場合*

   **%ovinstalldir%bin\win64\opcagt -kill**

   *それ以外の Windows ノードの場合*

   **%ovinstalldir%bin\opcagt -kill**

2. 次のコマンドを実行して、デフォルト以外のユーザーで実行されるようにエージェントを設定します。

   **cscript "%ovinstalldir%bin\ovswitchuser.vbs" -existinguser** <*DOMAIN\USER*> **-existinggroup** <*GROUP*> **-passwd** <*PASSWORD*>

   ここで、

   <*ドメイン\ユーザー*> はドメインとユーザー名です。

   <*GROUP*> はユーザーが属するグループの名前です (AgentGroup など)。

   <*パスワード*> はユーザーのパスワードです。

   **注:** このコマンドは、基本のエージェント機能に必要なユーザー権限を、個人ユーザーにではなくグループ レベルで割り当てます。そのため、使用するグループの選択には注意してください。このエージェント ユーザーのための新しいグループを作成し、エージェント ユーザーをメンバーとして追加することをお勧めします。

3. 次のコマンドを実行して、必要なアクセス権限をデフォルト以外のユーザーに設定します。

   **cscript %ovinstalldir%lbin\xpl\ovsetscmpermissions.vbs -user** <*User_Name*> **-f**

   ここで、<*User_Name*> はデフォルト以外のユーザーの名前です。

4. 次のいずれかのコマンドを実行します。

   ローカル システム以外のアカウントでエージェントのすべてのコンポーネントを実行するには

   *Windows 64 ビット ノードの場合*

   **%OvInstallDir%bin\win64\ovconfchg -ns eaagt -set NPU**

   *それ以外のすべての Windows ノードの場合*

   **%OvInstallDir%bin\ovconfchg -ns eaagt -set NPU**

   ローカル システム以外のアカウントで運用監視コンポーネント のみを実行するには

   *Windows 64 ビット ノードの場合*

   **%OvInstallDir%bin\win64\ovconfchg -ns eaagt -set MIXED**

   *それ以外のすべての Windows ノードの場合*

   **%OvInstallDir%bin\ovconfchg -ns eaagt -set MIXED**

5. 以下のコマンドを実行します。

   *Windows 64 ビット ノードの場合*

   **%OvInstallDir%bin\win64\ovconfchg -ns ctrl.sudo -set OV_SUDO_USER** <*User_Name*>

   **%OvInstallDir%bin\win64\ovconfchg -ns ctrl.sudo -set OV_SUDO_GROUP** <*Group_Name*>

   **%OvInstallDir%bin\win64\ovconfchg -ns eaagt -set OPC_PROC_ALWAYS_INTERACTIVE NEVER**

   *それ以外のすべての Windows ノードの場合*

   **%OvInstallDir%bin\ovconfchg -ns ctrl.sudo -set OV_SUDO_USER** <*User_Name*>

   **%OvInstallDir%bin\ovconfchg -ns ctrl.sudo -set OV_SUDO_GROUP** <*Group_Name*>

   **%OvInstallDir%bin\ovconfchg -ns eaagt -set OPC_PROC_ALWAYS_INTERACTIVE NEVER**

   ここで、<*User_Name*> はデフォルト以外のユーザーの名前であり、<*Group_Name*> はデフォルト以外のユーザーが属するグループです。

6. 権限のない操作モードを選択した場合、次のコマンドを実行します。

   *Windows 64 ビット ノードの場合*

   **%OvInstallDir%bin\win64\ovconfchg -ns eaagt -set OPC_RPC_ONLY TRUE**

   *それ以外のすべての Windows ノードの場合*

   **%OvInstallDir%bin\ovconfchg -ns eaagt -set OPC_RPC_ONLY TRUE**

7. 混合操作モードを選択した場合、次のコマンドを実行します。

   *Windows 64 ビット ノードの場合*

   **%OvInstallDir%bin\win64\ovconfchg -ns eaagt -set NPU_TASK_SET EVENT_ACTION**

   *それ以外のすべての Windows ノードの場合*

   **%OvInstallDir%bin\ovconfchg -ns eaagt -set NPU_TASK_SET EVENT_ACTION**

8. 次のコマンドを実行します。

   > **注:** この手順は UNIX/Linux ノードに必要であり、Windows ノードでは必須ではありません。ただし、Windows ノードでもこの手順を完了することをお勧めします。

   *Windows 64 ビット ノードの場合*

   **%OvInstallDir%bin\win64\ovconfchg -ns eaagt -set SNMP_TRAP_PORT** <*SNMP_Port*>

   **%OvInstallDir%bin\win64\ovconfchg -ns bbc.cb -set SERVER_PORT** <*Comm_Port*>

   *それ以外のすべての Windows ノードの場合*

   **%OvInstallDir%bin\ovconfchg -ns eaagt -set SNMP_TRAP_PORT** <*SNMP_Port*>

   **%OvInstallDir%bin\ovconfchg -ns bbc.cb -set SERVER_PORT** <*Comm_Port*>

   上記コマンドの説明:

<*Comm_Port*> は選択した通信ポート番号です。

<*SNMP_Port*> は HP Operations エージェント で SNMP トラップを受け取るポートです。

これらのポートは 1024 より大きい番号である必要があります。

9. エージェントを再起動します。

   *Windows 64 ビット ノードの場合*

   **%ovinstalldir%bin\win64\opcagt -start**

   *それ以外のすべての Windows ノードの場合*

   **%ovinstalldir%bin\opcagt -start**

## UNIX/Linux でのデフォルト ユーザーの変更

インストール時に、デフォルト以外のユーザーでエージェントを実行するように設定できない場合は (「インストール中のエージェント ユーザーの設定」(62ページ) を参照)、*非アクティブ*モードでインストールすることをお勧めします。詳細については、『*HP Operations エージェント インストール ガイド*』を参照してください。

デフォルト以外のユーザーを使用してエージェントを設定するには、次の方法を使用できます。

- プロファイル ファイルの使用
- ovswitchuser コマンドの使用

### プロファイル ファイルの使用

プロファイル ファイルを使ってデフォルト エージェント ユーザーを変更するには、次の手順を実行します。

1. エージェントをインストールするシステムで新しいファイルを作成して、そのファイルをテキスト エディタで開きます。
2. 次の文を指定して、エージェントの操作モードを指定します。

   ローカル システム以外のアカウントでエージェントを実行するには、次のように入力します。

   **set eaagt:MODE= NPU**

   ローカル システム以外のアカウントで運用監視コンポーネント のみを実行するには、次のように入力します。

   **set eaagt:MODE= MIXED**

3. 権限のないモードを選択した場合 (前の手順で `set eaagt:MODE=NPU` と入力した場合)、次の文を入力します。

   **set eaagt:OPC_RPC_ONLY=TRUE**

4. 権限のないモードを選択した場合 (前の手順で `set eaagt:MODE=NPU` と入力した場合)、次の文を入力します。

   **set eaagt:SNMP_TRAP_PORT=**<*SNMP_port_number*>

   **set bbc.cb:SERVER_PORT=**<*Comm_port_number*>

> **注 :** これは UNIX/Linux ノードには必須です。UNIX/Linux では root 以外のユーザーには 1024 未満のポートにアクセスできる権限がないためです。

上記コマンドの説明 :

*<Comm_Port>* は選択した通信ポート番号です。

*<SNMP_Port>* は HP Operations エージェント で SNMP トラップを受け取るポートです。

これらのポートは 1024 より大きい番号である必要があります。

5. 混合モード (手順 2 で `set eaagt:MODE=MIXED` と入力した場合 )、次の文を入力します。

   **set eaagt:NPU_TASK_SET=EVENT_ACTION**

6. 次の文を入力します。

   **set ctrl.sudo:OV_SUDO_USER=***<User_Name>*

   **set ctrl.sudo:OV_SUDO_GROUP=***<Group_Name>*

   ここで、*<User_Name>* はデフォルト以外のユーザーの名前であり、*<Group_Name>* はデフォルト以外のユーザーが属するグループです。

7. システムのローカルディレクトリにファイルを保存します。

8. プロファイルファイルで指定したユーザーで実行されるようにエージェントを再設定します。
   a. ノードの以下の場所に移動します。

      `/opt/OV/bin/OpC/install`

   b. 次のコマンドを実行します。

      **./oainstall.sh -a -configure -agent_profile** *<path>/<profile_file>*

      ここで、*<profile_file>* はプロファイルファイルの名前であり、*<path>* はプロファイルファイルへの完全パスです。

## もう一方の方法 : ovswitchuser コマンドの使用

プロファイルファイルを使用しない場合は、以下の手順に従います。

> **注 :** 設定手順としては、プロファイルファイルでの設定の方をお勧めします。

1. 以下のディレクトリに移動します。

   *HP-UX、Linux、Solaris の場合*

   `/opt/OV/bin`

   *AIX の場合*

   `/usr/lpp/OV/bin`

2. エージェントを停止します。

   **./opcagt -kill**

3. 次のコマンドを実行して、デフォルト以外のユーザーで実行されるようにエージェントを設定します。

   **./ovswitchuser.sh -existinguser** <*User_Name*> **-existinggroup** <*Group_Name*>

   ここで、

   <*User_Name*> はエージェントを実行するユーザーの名前です。

   <*Group_Name*> はユーザーが属するグループの名前です (AgentGroup など)。このコマンドを実行すると、このグループには、エージェント データ ディレクトリ内のすべてのファイルに対するフルコントロールが付与されます。また、すべてのインストール済みパッケージへのフルコントロールも付与されます。以前にこのコマンドを実行して異なるグループを指定している場合、このコマンドを実行すると、以前のグループのファイルのコントロールは削除されます。

   グループ ID フラグがエージェントのデータ ディレクトリに設定されます。このフラグは、指定するグループが、エージェントのベース ディレクトリ内のすべての新しいファイルとサブディレクトリも所有することを示します。

   > **注:** このコマンドは、基本のエージェント機能に必要なユーザー権限を、個人ユーザーにではなくグループ レベルで割り当てます。そのため、使用するグループの選択には注意してください。このエージェント ユーザーのための新しいグループを作成し、エージェント ユーザーをメンバーとして追加することをお勧めします。

4. 次のいずれかのコマンドを実行します。

   root 以外のユーザーでエージェントのすべてのコンポーネントを実行するには

   **./ovconfchg -ns eaagt -set NPU**

   root 以外のユーザーで運用監視コンポーネント のみを実行するには

   **./ovconfchg -ns eaagt -set MIXED**

5. 次のコマンドを実行します。

   > **注:** これは UNIX/Linux ノードには必須です。UNIX/Linux では root 以外のユーザーには 1024 未満のポートにアクセスできる権限がないためです。

   **./ovconfchg -ns eaagt -set SNMP_TRAP_PORT** <*SNMP_Port*>

   **./ovconfchg -ns bbc.cb -set SERVER_PORT** <*Comm_Port*>

   上記コマンドの説明:

   <*Comm_Port*> は選択した通信ポート番号です。

   <*SNMP_Port*> は HP Operations エージェント で SNMP トラップを受け取るポートです。

   これらのポートは 1024 より大きい番号である必要があります。

6. 以下のコマンドを実行して、エージェントを起動します。

   **./opcagt -start**

エージェントが root 以外のユーザーで実行されるように設定すると、**System.txt** ファイルに以下のエラー メッセージが表示されることがあります。

`ovbbccb (22461/1):(bbc-188)` 現行プロセスのルートディレクトリを変更できません。

このエラーは無視してください。

## コマンドのデフォルト ユーザーの変更

デフォルトでは、エージェントは、現在エージェント自体を実行しているユーザー アカウントで自動コマンドまたはオペレータ起動コマンドを開始します。ただし、別のユーザー アカウントでコマンドを開始するように HP Operations エージェントを設定できます。そのためには、ノードの eaagt 名前空間の OVO_STD_USER パラメータを設定する必要があります。このパラメータを設定する手順は次のとおりです。

- HP Operations エージェントのインストール時にデフォルトの値を設定します。この方法は、多数のノードでユーザーを設定する必要がある場合にお勧めします。ノードを作成および移行する前に、インストールのデフォルトを計画して設定する必要があります。
- コマンド プロンプトで ovconfchg または ovconfpar を使用します。
- *<ユーザー>*|*<暗号化されたパスワード>* の形式で OVO_STD_USER の値を指定します。

  *<ユーザー>* をユーザー名に置き換えます。ドメイン ユーザーの場合、ドメイン名とユーザー名を指定します (EXAMPLE\AgentUser など)。ローカル ユーザーの場合、名前のみを指定します (AgentUser など)。

  *<暗号化されたパスワード>* を **opcpwcrpt** *<パスワード>* の出力結果に置き換えます。このコマンドは、管理サーバーのコマンド プロンプトから実行できます。

ツールを設定または起動する場合は、OVO_STD_USER を使用することもできます。ユーザー名を $OVO_STD_USER とし、パスワードは空白のままとします。

コマンドやツールを正しく実行するための適切な権限がユーザー アカウント割り当てられているかどうかをテストする必要があります。

> **注 :** OVO_STD_USER でエージェントがコマンドまたはツールの開始に失敗する場合は、エージェントを現在実行しているユーザー アカウントでコマンドまたはツールが実行されている可能性があります。誤ったユーザーまたはパスワードを指定した場合などに、この現象が発生します。

# セキュリティ コンポーネント変数の設定

HP Operations エージェントをインストールするとき、セキュリティ コンポーネントである OvSecCm は、安全な通信に使用される長さが 2048 の RSA 鍵のペアをデフォルトで作成します。RSA 鍵の長さを変更するには、ASYMMETRIC_KEY_LENGTH の設定変数を更新します。

設定変数を更新するには、以下のコマンドを実行します。

*ovconfchange -edit*

管理サーバーで設定を更新するには、MigrateAsymKey ツールを使用できます。このツールを使用すると、新しい CA 証明書を作成したり、鍵ペアを更新したり、新しい CA 証明書をノードに追加したりできます。

管理サーバーで設定を更新するには、以下の手順を実行します。

1. 管理サーバーで以下の場所に移動します。
   - **Windows の場合**

%ovinstalldir%\lbin\seccs\install\MigrateAsymKey.vbs

- **UNIX の場合**

%ovinstalldir%/lbin/seccs/install/MigrateAsymKey.sh

- **AIX の場合**

%ovinstalldir%/lbin/seccs/install/MigrateAsymKey.sh

2. 移行ツールを実行します。

**Windows の場合**

```
cscript MigrateAsymKey.vbs -createCAcert
```

**UNIX の場合**

```
./MigrateAsymKey.sh -createCAcert
```

ASYMMETRIC_KEY_LENGTH の設定変数に設定された新しい鍵長の値に応じて、コマンドにより新しい CA 証明書が作成されます。

3. サーバーで管理されているすべてのノードで信頼のおける証明書を更新するには、次のコマンドを実行します。

*ovcert –updatetrusted*

> **注 :** 信頼済み証明書がノードで更新されていることを確認するには、次のコマンドを実行する必要があります。コマンドは管理サーバーからリモートで実行することも、ノードで実行することもできます。

4. サーバーに新しいノード証明書を作成するには、以下のコマンドを実行します。

**Windows の場合**

*cscript MigrateAsymKey.vbs -createNodecert*

**UNIX の場合**

*./MigrateAsymKey.sh -createNodecert*

コマンドにより、新しい RSA 鍵ペアを持つ新しいノード証明書が作成されます。

> **注 :** ovcert - certinfo <certificate_alias> のコマンドを実行すると、更新された鍵を確認できます。
>
> 鍵の長さは ASYMMETRIC_KEY_LENGTH で更新されます。

# vMA ノードの監視用 viserver の設定

*vMA ノードを監視しない場合は、このセクションをスキップしてください。*

viserver デーモンは、**viserver.properties** と **VILog4j.xml** の2 つの設定ファイルの内容を変更することによって設定できます。これらのファイルは **/var/opt/perf** にあります。

# viserver.properties

このファイルには次のパラメータが含まれます。

port

hosts

instance

jvmArgs

log4Interval

**viserver.properties** ファイルの設定を変更する場合、viserver を再起動する必要があります。viserver を再起動した後にのみ、新しい設定が有効になります。

## port

port パラメータは、viserver およびクライアントが通信するループバックポートです。port パラメータは編集できません。このパラメータの値は **viserver** を再起動すると変わります。

## hosts

hosts パラメータでは、viserver デーモンがサポートできるホストの数を定義します。デフォルト値は 20 です。

環境にそれ以上のホストがある場合、このパラメータを編集して、必要な設定を指定できます。指定したホスト数のデータを HP Operations エージェントで収集できない場合は、vifp ターゲットを少なくする必要があります。

## instance

instance パラメータでは、viserver でサポートできるインスタンスの数を定義します。デフォルト値は 200 です。

環境にそれ以上のインスタンスがある場合、このパラメータを編集して、必要な設定を指定できます。指定したインスタンス数のデータを HP Operations エージェントで収集できない場合は、vifp ターゲットを少なくする必要があります。

## jvmArgs

jvmArgs パラメータを使うと、環境に jvm 引数を追加したり、必要に応じて jvm を変更したりできます。

jvmArgs のデフォルト設定は以下のようになります。

```
jvmArgs=-Xms512m -Xmx1024m -classpath
/opt/perf/bin/java/activation.jar\:/opt/perf/bin/java/axis-
```

```
ant.jar\:/opt/perf/bin/java/axis.jar\:/opt/perf/bin/java/commons-
discovery-0.2.jar\:/opt/perf/bin/java/commons-logging-
1.0.4.jar\:/opt/perf/bin/java/jaxrpc.jar\:/opt/perf/bin/java/log4j-
1.2.8.jar\:/opt/perf/bin/java/mailapi.jar\:/opt/perf/bin/java/saaj.jar\:/opt/perf/bin
1.5.1.jar\:/opt/vmware/vma/lib64/vmatargetlib25.jar\:/opt/vmware/vma/lib64/vifplib25.
 com.hp.perfagent.VIdaemon
```

### log4jInterval

log4jInterval パラメータでは、viserver が **VILog4j.xml** ファイルの変更をチェックする間隔を指定します。デフォルト値は 60000 ミリ秒 (1 分) です。この値は必要に応じて変更できます。

## VILog4j.xml

viserver は **VIlog4j.xml** ファイル (**/var/opt/perf** に配置) を使用して、**status.viserver** ファイルにログステータス情報を記録します。**log4j.dtd** ファイル (**/var/opt/perf** に配置) は **VIlog4j.xml** ファイルのテンプレートを定義します。**VIlog4j.xml** ファイルの設定を変更できますが、指定した時間経過後に変更が有効になります。指定する時間の値は log4jInterval パラメータで定義します。

| | |
|---|---|
| | XML ファイルには、正しくロギングを行うのに必要な要素があります。これらの要素を変更したり削除したりしないでください。変更するのは、com.hp.perfagent ロガーのレベルのみとすることをお勧めします。 |

XML ファイルは主に以下の要素で構成されます。

```
appender
```

```
logger
```

変更できるのは、
[<logger name="com.hp.perfagent"> ... </logger>] 要素内の以下の項目だけです。

```
<level value = "info"/>
```

このエンティティは、**status.viserver** ファイルのロギングレベルを決定します。以下のデフォルト以外の設定のいずれかに value を設定できます。

fatal: この設定を使用すると最小限の情報を記録します。

debug: この設定を使用するとデバッグ情報を記録します。

| | |
|---|---|
| | debug 設定はトラブルシューティングを行う場合にのみ使用してください。 |

# Windows でのアプリケーションおよびサービス ログの監視

HP Operations エージェントの Logfile Encapsulator コンポーネントを使うと、Windows イベント ログを監視できます。Windows イベント ログ ポリシーは、選択した Windows イベント ログを監視するようにエージェントを設定するために役立ちます。

次の Windows のバージョンには、イベント ログ—アプリケーションおよびサービス ログの新しいカテゴリが用意されています。

- Windows Vista
- Windows Server 2008
- Windows Server 2008 R2
- Windows 7

適 切 な Windows イベント ログ ポリシーを設 定 すると、HP Operations エージェント 11.10でこれらのアプリケーションおよびサービス ログを監 視 できます。

HP Operations エージェントは次 のタイプのイベント ログを監 視 できません。

- リモート システムで生 成 されたイベント (Windows のイベント サブスクリプション機 能 を使 用 して収 集 )
- 保 存 されたイベント ログ

次 のイベント レベルを持 つイベント :

- LOG_ALWAYS
- VERBOSE

## HPOM for Windows からのアプリケーション ログおよびサービス イベント ログの監 視

アプリケーションおよびサービス イベント ログを監 視 するために Windows のイベント ログ ポリシーを作 成 するには、以 下 の手 順 に従 います。

1. Windows イベント ログがある Windows ノード にログオンします。
2. Event Viewer ウィンド ウを開きます。
3. コンソール ツリーで、イベントを選 択します。詳 細 ペインにイベント ログの名 前 が表 示 されます ([ログ名 ] フィールド の横 )。

   | | |
   |---|---|
   | Log Name: | Microsoft-Windows-Bits-Client/Operational |
   | Source: | Bits-Client |
   | Event ID: | 306 |

   ログ ファイル の名 前 を詳 細 ペインに表 示 されているとおりにメモに残します。
4. HPOM for Windows コンソールを開きます。
5. コンソール ツリーの [タイプ別 にグループ化 されたエージェント ポリシー] で、[**Windows イベント ログ**] を右 クリックして、**[新 規 ] > [ポリシー]** をクリックします。
6. Windows イベント ログ ポリシー用 のポリシー エディタが開きます。
7. [セキュア] タブで Windows イベント ログの名 前 (手 順 3 で記 録 した名 前 ) を [イベント ログ名 ] フィールド に入 力します。

   Event log name* Microsoft-Windows-Bits-

8. HPOM for Windows オンライン ヘルプの手順に従って、他の詳細をポリシーに指定します。
9. ポリシーを保存します。
10. Windows ノードにポリシーを配布します。

## UNIX/Linux 9.1x 上の HPOM からのアプリケーション ログおよびサービス イベント ログの監視

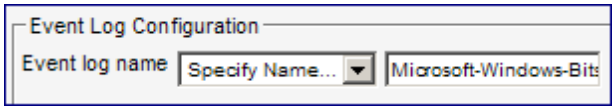
アプリケーションおよびサービス イベント ログを監視するために Windows のイベント ログ ポリシーを作成するには、以下の手順に従います。

1. Windows イベント ログがある Windows ノードにログオンします。
2. Event Viewer ウィンドウを開きます。
3. コンソール ツリーで、イベントを選択します。詳細ペインにイベント ログの名前が表示されます ([ログ名] フィールドの横)。

   | | |
   |---|---|
   | Log Name: | Microsoft-Windows-Bits-Client/Operational |
   | Source: | Bits-Client |
   | Event ID: | 306 |

   ログ ファイルの名前を詳細ペインに表示されているとおりにメモに残します。
4. HPOM for UNIX 管理 UI にログオンします。
5. **[OMU]** をクリックします。
6. **[参照] > [すべてのポリシー タイプ]** をクリックします。
7. [Windows のイベント ログ] をクリックします。「Windows_Event_Log」というポリシー タイプのページが開きます。
8. をクリックして、**[新規ポリシー]** をクリックします。「Windows_Event_Log Policy の追加」というページが開きます。

   [ソース] タブの [イベント ログ名] フィールドで、[名前を指定] を選択します。新しいテキスト ボックスが表示されます。Windows イベント ログの名前 (手順 3 で記録した名前) をテキスト ボックスに入力します。

   
   
9. HPOM for UNIX オンライン ヘルプの手順に従って、他の詳細をポリシーに指定してます。
10. ポリシーを保存します。
11. Windows ノードにポリシーを配布します。

## HPOM for UNIX 8.35 からのアプリケーション ログおよびサービス イベント ログの監視

アプリケーションおよびサービス イベント ログを監視するために Windows のイベント ログ ポリシーを作成するには、以下の手順に従います。

1. Windows イベント ログがある Windows ノードにログオンします。
2. Event Viewer ウィンドウを開きます。
3. コンソール ツリーで、イベントを選択します。詳細ペインにイベント ログの名前が表示されます ([ログ名] フィールドの横)。

| | |
|---|---|
| Log Name: | Microsoft-Windows-Bits-Client/Operational |
| Source: | Bits-Client |
| Event ID: | 306 |

| | |
|---|---|
| Log Name: | Microsoft-Windows-Bits-Client/Operational |
| Source: | Bits-Client |
| Event ID: | 306 |

4. HPOM for UNIX コンソールにログオンします。
5. [メッセージ ソース テンプレート] ウィンドウに移動し、新しいログファイル テンプレートを作成します。
6. [ログファイル] フィールド ([ログ ファイルの追加] ダイアログ ボックス) で、2 つの % 文字を入力して、2 つの % 文字の間に Windows イベント ログの名前 (手順 3 で記録した名前) を入力して、その後ろに _LOG のサフィックスを付けます。たとえば、イベント ログ名が *<event_log_name>* の場合、%*<event_log_name>*_LOG**%** と入力する必要があります。

**Logfile**

%Microsoft-Windows-Bits-Client/Operational_LOG%

7. HPOM for UNIX オンライン ヘルプの手順に従って、他の詳細をポリシーに指定してます。
8. ポリシーを保存します。
9. Windows ノードにポリシーを配布します。

## リモート ノードの監視

エージェント ソフトウェアがインストールされていないシステムで、SNMP トラップ、SNMP MIB データ、または WMI イベントを監視できます。

# utility プログラムの使い方

utility プログラムは、ログ ファイル、収集パラメータ (parm) ファイル、アラーム定義 (alarmdef) ファイルに関する情報を管理し、レポートするためのツールです。対話型モードまたはバッチ モードで utility プログラムを使用すると、次の作業を行うことができます。

- 生ログ ファイルまたは抽出ログ ファイルの走査と、以下の項目を含んだレポートの作成
  - 扱われている日時
  - scope コレクタを実行していない時間
  - scope パラメータ設定値の変更
  - システム構成の変更
  - ログ ファイル ディスク スペース
  - 収集パラメータ (parm) ファイルでのアプリケーションおよびプロセスの設定値の効果
- 生ログ ファイルのサイズ変更
- parm ファイル内の構文の警告やエラーの確認
- alarmdef ファイル内の構文の警告やエラーの確認
- アラーム定義と比較したログ ファイル データの処理と、履歴データ内のアラーム状態の検出

この章では次の内容について説明します。

- utility プログラムの実行
- 対話型モードの使い方
- utility のコマンド ライン インターフェイスの使い方
- utility 走査レポートの詳細

utility プログラムの各コマンドの詳細は、5 章「utility のコマンド」を参照してください。

# utility プログラムの実行

utility プログラムを実行するには、次の 3 つの方法があります。

- コマンド ライン モード — utility プログラムはコマンド ラインでコマンド オプションと引数を使用して制御します。
- 対話型モード — stdin が対話式の端末またはワークステーションに設定されたプログラムを実行するときに、対話型のコマンドとパラメータを入力します。
  使用経験があるユーザーの場合は、特定の作業に必要なコマンドだけを直ちに指定してもかまいません。初めて使用するユーザーの場合は、utility プログラムの guide コマンドを使用してコマンドの使い方のガイドを表示することをお勧めします。ガイド モードでは、タスクを実行するときに、オプション リストからの選択を促すプロンプトが表示されます。また、ガイド モードでは、各作業を行う対話型のコマンドが実行時にリストされるため、それらのコマンドの使い方を確認できます。ガイド モードは、随時終了し、再開始できます。
- バッチ モード — プログラムを実行し、対話型のコマンドとパラメータを含むファイルに stdin をリダイレクトできます。

コマンド ライン インターフェイスの構文は、その他のプログラムで使用する標準的な UNIX コマンド ライン インターフェイスと類似しています。この構文の詳細はこの章で説明します。

対話型モードとバッチ モードのコマンド構文は同一です。コマンドは、どのような順序で入力してもかまいません。コマンドに関連付けるパラメータがある場合、そのパラメータは対応するコマンドの直後に入力する必要があります。

パラメータには、必須のパラメータ (デフォルトなし) とオプションのパラメータ (デフォルトあり) の 2 種類があります。utility がこれらのパラメータを扱う方法は、実行モードによって異なります。

- 対話型モードでオプションのパラメータが欠落している場合は、プログラムがデフォルトの引数を表示します。ユーザーはその引数を確定するか、無効にする必要があります。必須のパラメータが欠落している場合は、プログラムが引数を入力するように促すプロンプトを表示します。
- バッチ モードでオプションのパラメータが欠落している場合、プログラムはデフォルト値を使用します。必須のパラメータが欠落している場合、プログラムは終了します。

対話型モードの場合とコマンド ラインおよびバッチ モードの場合では、エラーと欠落データが異なる方法で処理されます。対話型モードではデータの追加や間違いの訂正ができますが、コマンド ラインやバッチ モードではできません。

# 対話型モードの使い方

utility プログラムの対話型モードを使用するには、特定の作業を行うための一連のコマンドを発行する必要があります。

たとえば、今日記録されたデータにアラーム状態が存在しているかどうかをログ ファイルで確認する場合は、utility プログラムを呼び出した後に次のコマンドを発行します。

```
checkdef /var/opt/perf/alarmdef

 detail off

 start today-1

 analyze
```

checkdef コマンドが、alarmdef ファイル内のアラーム定義構文を確認し、analyze コマンドで使用するファイル名を設定して保存します。detail off コマンドを発行すると、analyze コマンドがアラームの要約だけを表示します。start today-1 コマンドは、昨日記録されたデータだけを分析するように指定します。analyze コマンドは、デフォルトの SCOPE データ ソース内にある生ログ ファイルを alarmdef ファイルと比較して分析します。

# 対話型モードとバッチ モードの使用例

次の例では、バッチ モードと対話型モードにおける utility プログラムの resize コマンドの機能の違いを示します。

resize コマンドでは、次の関数にパラメータを設定できます。

- サイズ変更するログ ファイルの種類
- 新しいファイルのサイズ
- ファイルに残す空きスペースの大きさ
- サイズ変更の実行を指定する動作

次の resize コマンドの例では、グローバル ログ ファイルのサイズを、45 日分の空きスペースを残して最大 120 日分のデータを格納できるサイズに変更します。コマンドとそのパラメータは次のようになります。

```
resize global days=120 empty=45 yes
```

このコマンドを対話式に入力しても、バッチ ジョブから入力しても、同じ結果が得られます。

最初のパラメータ (global) は、サイズを変更するログ ファイルを示しています。このパラメータを指定しないと、対話型モードとバッチ モードでは、次のような動作が行われます。

- バッチ モード — logfile パラメータにはデフォルトがないため、バッチ ジョブが終了します。
- 対話型モード — サイズ変更を完了するためにログ ファイルの種類を選択するように促すプロンプトが表示されます。

最後のパラメータ (yes) は、サイズ変更を無条件で実行することを示しています。

パラメータ yes を指定しないと、対話型モードとバッチ モードでは、次のような動作が行われます。

- バッチ モード — yes がデフォルトの動作であるためサイズ変更が続けられます。
- 対話型モード — サイズ変更を行う前に動作の指定を促すプロンプトが表示されます。

バッチ モードまたは対話型モードで resize コマンドを使用する前に、データ収集を終了する必要があります。詳細は、2 章の「データ収集の終了と再開始」を参照してください。

# utility のコマンド ライン インターフェイス

対話型モードとバッチ モードのコマンド構文のほかに、コマンド ライン インターフェイスを使用してもコマンド オプションとそれに関連する引数を utility プログラムに渡すことができます。utility プログラムをシェル スクリプトで簡単に呼び出せるようにし、UNIX のパイプへの入出力のリダイレクトを可能にすることで、コマンド ライン インターフェイスは一般的な UNIX 環境に適合します。

たとえば、コマンド ラインを使用して、前述の「対話型モードの使い方」に示されている例と同様の処理をするには、次のように入力します。

```
utility -xr global days=120 empty=45 yes
```

次の表で、コマンド ラインのオプションと引数を示します。それぞれのコマンドの説明は、5 章「utility のコマンド」に記載されています。

**表 2:コマンド ラインの引数**

| コマンド オプション | 引数 | | 説明 |
|---|---|---|---|
| -b | date | time | 分析または走査関数の開始日時を指定します(5 章の start コマンドの項を参照)。 |
| -e | date | time | 分析または走査関数の終了日時を指定します(5 章の stop コマンドの項を参照)。 |

| | | | |
|---|---|---|---|
| -l | logfile | | オープンするログ ファイルを指 定します(5 章 の logfile コマンド の項 を参 照 )。 |
| -f | listfile | | 出 力リスト ファイルを指 定します(5 章 の list コマンド の項 を参 照 )。 |
| -D | | | analyze、scan および parm ファイルの確 認 で詳 細 を有 効 にします(5 章 の detail コマンド の項 を参 照 )。 |
| -d | | | analyze および parm ファイルの確 認 で詳 細 を無 効 にしま す(5 章 の detail コマンド の項 を参 照 )。 |
| -v | | | コマンド ラインのコマンド を実 行 時 にエコーさせます。 |
| -xp | parmfile | | parm ファイルの構 文 を確 認 します(5 章 の parmfile コマンド の項 を参 照 )。 |
| -xc | alarmdef | | -xa (または analyze コマンド ) で使 用 する alarmdef ファ イルの構 文 を確 認 し、そのファイル名 を設 定 します(5 章 の checkdef コマンド の項 を参 照 )。 |
| -xa | | | alarmdef ファイルと比 較 してログ ファイルを分 析 します(5 章 の analyze コマンド の項 を参 照 )。 |
| -xs | logfile | | ログ ファイルを走 査 し、レポートを作 成 します(5 章 の scan コ マンド の項 を参 照 )。 |
| -xr | global<br>application<br>process<br>device<br>transaction<br>ls<br>EMPTY=nnn<br>SPACE=nnn | SIZE=nnn<br>DAYS=nnn<br><br><br>あり<br>なし<br>MAYBE | ログ ファイルのサイズを変 更 します(5 章 の resize コマンド の 項 を参 照 )。 |
| -? ま たは ? | | | コマンド ライン構 文 を表 示 します。 |

## コマンド ライン インターフェイスの使 用 例

コマンド ラインにコマンド オプションと引 数 を入 力 するときには、次 の規 則 が適 用 されます。

エラーと欠 落 データは、対 応 するバッチ モード コマンド とまったく同じように処 理 されます。すなわち、欠 落 データには可 能 な場 合 デフォルト が使 用 され、エラーがあるとプログラムは直 ちに終 了 します。

コマンド およびコマンド の実 行 結 果 のエコーは無 効 に設 定 されます。Utility はその stdin ファイル から読 み込 みを行 いません。utility はコマンド ラインに指 定 されている動 作 を実 行 してから終 了 しま す。

```
utility -xp -d -xs
```

これは、次のように解釈します。

| | |
|---|---|
| -xp | デフォルトの parm ファイルの構文を確認します。 |
| -d | 走査レポートで詳細を無効にします。 |
| -xs | 走査を実行します。ログファイルが指定されていないため、デフォルトのログファイルが走査されます。 |

# utility 走査レポートの詳細

utility プログラムの scan コマンドは、ログファイルを読み込み、その内容に関するレポートを作成します。レポートの内容は、scan コマンドの前に発行されたコマンドによって異なります( 詳細は、5 章の「utility のコマンド」の「scan」コマンドの項を参照してください)。

次の表は、すべての走査レポートや、detail on コマンドが scan コマンドと共に使用されたとき (デフォルト) にのみ作成されるレポートに含まれる情報をまとめたものです。

表 3:走査レポートに含まれる情報

| 初期値 | |
|---|---|
| 初期 parm ファイルのグローバル情報およびシステム構成情報 | detail on が指定された場合にのみ出力されます。 |
| 初期 parm ファイルのアプリケーション定義 | detail on が指定された場合にのみ出力されます。 |
| **年代順の詳細** | |
| parm ファイルのグローバル変更内容 | detail on が指定された場合にのみ出力されます。 |
| parm ファイルのアプリケーション変更内容 | detail on が指定された場合にのみ出力されます。 |
| コレクタのオフタイム通知 | detail on が指定された場合にのみ出力されます。 |
| アプリケーション固有の要約レポート | detail on が指定された場合にのみ出力されます。 |
| **要約** | |
| プロセス要約レポート | プロセスデータが走査された場合、常に出力されます。 |
| コレクタの適用範囲の要約 | 常に出力されます。 |
| ログファイルの内容の要約 | 常に出力されます。スペースおよび日付を含みます。 |
| ログファイルの空きスペースの要約 | 常に出力されます。 |

# 走査レポート情報

utility 走査レポートの情報は、次の3つの種類に分類されます。

- 初期値
- 年代順の詳細
- 要約

## 初期値

この項では、次の初期値について説明します。

- 初期 parm ファイルのグローバル情報
- 初期 parm ファイルのアプリケーション定義

### 初期 parm ファイルのグローバル情報

このレポートを生成するには、scan コマンドをデフォルトの detail on と共に使用します。

このレポートは、ログファイル内で最も古いグローバルレコードの時刻における parm ファイルの構成の設定をリストします。その後のグローバル情報変更通知は、このレポートに含まれる値に基づいて行われます。特定のパラメータについて変更通知が存在しない場合、走査期間中そのパラメータが元の設定値を維持していたことを意味します。

次の例は、parm ファイルの内容をリストするレポートの一部です。

```
06/03/99 15:28 System ID="Homer"

scopeux/UX A.10.00 SAMPLE INTERVAL = 300,300,60 Seconds, Log version=D

Configuration: 9000/855, O/S A.10.00 CPUs=1

Logging Global Process records

        Device= Disk FileSys records

Thresholds: CPU= 10.00%, Disk=10.0/sec, First=5.0 sec, Resp=30.0 sec,

            Trans=100 Nonew=FALSE, Nokilled=FALSE, Shortlived=FALSE
            (<1 sec)

HP-UX Parms: Buffer Cache Size = 16384KB, NPROC = 532

Wait Thresholds: CPU=100.00%, Memory=100.00%

Impede=100.00%

Memory: Physical = 84.0 MB, Swap = 124304.0 MB, Available to users =
66.5 MB.There are 2 LAN interfaces:0, 1.

06/03/99 15:28 There are 2 disk devices:
   Disk #1976            = "/dev/hdisk0"
   Disk #1987            = "/dev/hdisk1"
```

最初の行にリストされている日時は、グローバルログファイル内での最初の日時であり、scope の開始時刻を示しています。グローバルログファイルからデータレコードがロールアウトされている可能性もあるため、このレポートに示されている日時が、必ずしもそのログファイルの最初のグローバルレコードであるとは限りません。

# 初期 parm ファイルのアプリケーション定義

このレポートを生成するには、scan コマンドをデフォルトの detail on と共に使用し、ログファイルにアプリケーション データを記録します。

このレポートは、最初のアプリケーション レコードがログファイルにリストされた時点の各アプリケーションの名前と定義をリストします。アプリケーションの追加または削除に関する通知は、この初期アプリケーション リストに基づいて行われます。たとえば、次のようになります。

```
06/01/99 08:39 Application(1) = "other"

Comment=all processes not in user-defined applications

06/01/99 08:39 Application(2) = "Real_TimeSystem"

 Priority range = 0-127

06/01/99 08:39 Application(3) = "Prog_Development"

 File=vi,ed,sed,xdb,ld,lint,cc,ccom,pc,pascomp
```

走査中に、追加または削除されたアプリケーションが通知されます。追加と削除は、古い名前と新たに記録されたアプリケーション名のスペルと大文字小文字を比較することによって判断されます。アプリケーションの定義の変更は検出されません。新しい名前のアプリケーションが検出された場合、そのアプリケーションが新しい定義と共にリストされます。
このレコードで示される日時は、現在ログファイルにある最初のアプリケーション レコードが記録される以前に scope が最後に開始された日時です。

# 年代順の詳細

この項では、次の年代順の詳細について説明します。

- parm ファイルのグローバル変更通知
- parm ファイルのアプリケーション追加通知および削除通知
- scope のオフタイム通知
- アプリケーション固有の要約レポート

## parm ファイルのグローバル変更通知

このレポートを生成するには、scan コマンドをデフォルトの detail on と共に使用します。

このレポートは、scope が開始したことを示すレコードが発見されたときに生成されます。

次の例では、2 つのディスクドライブがシステムに追加されたときに行われる変更通知を示します。

```
03/13/99 17:30 The number of disk drives changed from 9 to 11
03/13/99 17:30  New disk device scsi-4 = "c4d0s*"
03/13/99 17:30  New disk device scsi-3 = "c3d0s*"
```

## parm ファイルのアプリケーション追加通知および削除通知

このレポートを生成するには、scan コマンドをデフォルトの detail on と共に使用し、ログファイルにアプリケーション データを記録します。

ユーザー定義のアプリケーションは、scope を開始するたびに追加または削除できます。最後のアプリケーション セットに一致しないアプリケーション名が発見された場合、アプリケーションの追加、削除、変更を示す通知が出力されます。アプリケーション名が変更されていない場合は出力されません。

次の例は、新しいアプリケーションを開始したことを示しています。

```
03/13/99 17:30 Application 4 "Accounting_Users_1" was added
User=ted,rebecca,test*,mark,gene
```

> アプリケーションの定義の変更は確認されません。アプリケーションの定義は、アプリケーション名が変更されたときにリストされますが、名前の変更を伴わない既存のアプリケーション定義に対する変更は検出されません。

## scope のオフタイム通知

このレポートを生成するには、scan コマンドをデフォルトの detail on と共に使用します。

抽出ファイルに要約情報のみが含まれている場合、時刻の端数が切り捨てられ、最も近い時刻で示されます。たとえば、次のようになります。

```
06/03/99 11:00 - 06/03/99 12:34 collector off (01:34:04)
```

最初の日時 (06/03/99 11:00) は、scope を再開始する前のログファイルで最後の有効なデータレコードを示しています。2 番目の日時 (06/03/99 12:34) は、scope を再開始した日時を示しています。

最後のフィールド (かっこ内) は、scope を実行していない時間の長さを示します。フォーマットは *ddd/hh:mm:ss* です。*ddd* は日数、*hh:mm:ss* は、それぞれ時間、分、秒数です。左側の 0 は削除されます。

この例では、scope が、1999 年 6 月 3 日の 11:00 am から 12:34 pm まで停止していました。要約情報は、データが 1 時間 34 分の間収集されていないことを示しています。

## アプリケーション固有の要約レポート

このレポートを生成するには、scan コマンドをデフォルトの detail on と共に使用し、ログファイルにアプリケーション データを記録します。

このレポートは、アプリケーションを定義するときに役立ちます。蓄積しているシステム リソースが多すぎるアプリケーションや少なすぎるアプリケーション、他のアプリケーションと統合できるアプリケーションを特

定するときには、このレポートを使用してください。蓄積しているシステムリソースが多すぎるアプリケーションを、複数のアプリケーションに分割すると、パフォーマンスが向上します。

アプリケーションの定義は、システムパフォーマンスの調整について判断しやすい方法で行う必要があります。システムリソースがどのアプリケーションにも均等に蓄積することはあまりありません。

アプリケーションの定義が変更されるたびにアプリケーション固有の要約レポートが生成されるため、ユーザーはアプリケーションの定義の変更前と変更後のデータにアクセスできます。

すべてのアプリケーションに関する最終レポートが生成されます。このレポートに含まれる期間は、最後のレポート以降の期間であり、そのログファイルに含まれるすべての期間ではありません。たとえば、次のようになります。

```
                                PERCENT OF TOTAL
Application            Records   CPU    DISK   TRANS
------------------- --------- ------ ------- ------
OTHER                    22385  45.7%  20.9%  63.0%
Resource_Sharing          7531   6.0%   2.2%  17.1%
SPOOLING                 13813   2.4%   0.3%   0.0%
ON_LINE_COMPILES         13119   2.9%   1.7%   0.1%
BATCH_COMPILES            8429   2.9%   0.1%   2.2%
ORDER_ENTRY                387   0.1%   0.0%   0.0%
ELECTRONIC_MAIL           6251   3.8%   1.3%   9.6%
PROGRAM_DEVELOPMENT       3141   9.1%   2.4%   0.6%
RESEARCH_DEPARMENT        3968   8.7%   2.0%   6.0%
BILL_OF_MATERIALS          336   0.6%   1.5%   0.1%
FINANCIALS                1080   5.0%   1.5%   0.5%
MARKETING_DEPT            2712  12.9%  67.3%   0.0%
GAMES                      103   0.1%   0.0%   0.0%
-------------------- --------- ------ ------ ------
All user applications      73.1%   54.3%  79.1%  37.0%
```

# 要約

この項では、次の要約について説明します。

- プロセスの記録理由の要約
- 実際の走査の開始日時および終了日時
- アプリケーションの全体的な要約
- scope の適用範囲の要約
- ログファイルの内容の要約
- ログファイルの空きスペースの要約

## プロセスの記録理由の要約

このレポートを生成するには、ログファイルにプロセスデータを記録する必要があります。

このレポートは、scope の対象プロセスのしきい値を設定するときに役立ちます。このレポートは、プロセスが対象プロセスと見なされて記録された理由と、各条件を満たして記録されたプロセスの総数をリストします。

次のページの例は、プロセスの記録理由の要約レポートを示しています。

```
Process Summary Report: 04/13/99 3:32 PM to 05/04/99 6:36 PM

There were 93.8 hours of process data

Process records were logged for the following reasons:

Log Reason        Records   Percent   Recs/hr

---------------   -------   -------   -------

New Processes       17619     53.9%      44.7

Killed Processes    16047     49.1%      40.7

CPU Threshold        3169      9.7%       8.0

Disk Threshold       1093      3.3%       2.8
```

メモ : 1 つのプロセスは 1 度に複数の理由を記録できます。記録はカウントされますが、プロセス記録の 100% を超えることはありません。

detail on コマンドが発行された場合は、しきい値が変更されるたびにこのレポートが生成されるため、その変更の影響を評価できます。各レポートは、最後のレポート以降の期間をカバーしています。走査終了時に生成される最終レポートは、最後のレポート以降の期間をカバーしています。

detail off コマンドが発行された場合、走査期間全体をカバーしたレポートが 1 つのみ生成されます。

scope で記録するプロセスデータの量は、parm ファイルの threshold パラメータを変更し、ほとんどのプロセスログレコードを生成する条件となるしきい値を上げると減少します。記録するデータの量を増加するには、対象領域のしきい値を下げてください。

前の例では、CPU しきい値を上げるか、nonew しきい値を設定することで、プロセスデータに使用するディスクスペースの量を減らせます (記録される情報量は少なくなります)。

## 走査の開始および終了

この要約レポートは、有効なデータが走査された場合に出力されます。実際の走査の開始日時および終了日時が示されます。たとえば、次のようになります。

```
Scan started on        03/03/99 12:40 PM

Scan stopped on        03/11/99  1:25 PM
```

## アプリケーションの全体的な要約

このレポートを生成するには、ログファイルにアプリケーション データを記録する必要があります。

このレポートは、ユーザー定義のアプリケーションで蓄積されたシステム アクティビティの量に関する総合的な指標であり、その他のアプリケーションに関する情報は含まれていません。ユーザー アプリケー

ションが大量のクリティカルリソースを取り込んでいない場合、ユーザーアプリケーションに含められるプロセスを特定するためにプロセスデータの走査を検討する必要があります。

たとえば、次のようになります。

```
OVERALL, USER DEFINED APPLICATIONS ACCOUNT FOR

    82534 OUT OF    112355 RECORDS     ( 73.5%)

    218.2 OUT OF     619.4 CPU HOURS   ( 35.2%)

     24.4 OUT OF      31.8 M DISC IOS  ( 76.8%)

      0.2 OUT OF       0.6 M TRANS     ( 27.3%)
```

## コレクタの適用範囲の要約

このレポートは、有効なグローバルデータまたはアプリケーションデータが走査されたときに出力されます。このレポートは、システムの活動状況の収集に scope がどの程度有効に使用されたかを示します。次の例に示されるように、scope の停止時間の割合が高い場合は、scope の開始および終了に関する操作手順を見直す必要があります。

```
The total time covered was         108/16:14:51 out of 128/00:45:02

Time lost when collector was off   19/08:30:11  15.12%

The scopeux collector was started      45 times
```

グローバル詳細データが走査に含まれている場合、このレポートはより完全なものになります。入手できるデータが要約データだけである場合は、scope の開始時刻および終了時刻を最も近い時刻に決定します(この場合は、適切な警告メッセージがレポートと共に出力されます)。

カバーされた合計時間は、記録データから得られるすべてのインターバルを加算することで決定されます。「out of」時間値は、終了日時から開始日時を差し引くことで算出されます。これは、記録が可能であった合計時間を表しています。「Time lost when collector was off」値は、合計時間からカバーされた時間を差し引いた時間です。

上記の3つの時刻を表すフォーマットは次のとおりです。

*ddd/hh:mm:ss*

*ddd* は日数、*hh:mm:ss* はそれぞれ時間、分、秒数です。

上記の例では、収集された合計時間は、108 日と 16 時間 14 分 51 秒でした。

## ログファイルの内容の要約

ログファイルの内容の要約は、有効なデータが走査された場合に出力され、ログファイルのスペースとカバーされた日付が示されます。この要約は、resize コマンドでログファイルのサイズを変更するときに役立ちます。

```
--------Total-------  -----Each Full Day-----  -------Dates--------
Full

Type          Records  MBytes  Records  MBytes  Start         Finish
Days
```

```
Global          1376     0.27     288.9   0.057  05/23/99 to 05/28/99
4.8

Application     6931     0.72    1455.0   0.152  05/23/99 to 05/28/99
4.8

Process         7318     1.14    1533.6   0.239  05/23/99 to 05/28/99
4.8

Disk            2748     0.07     567.6   0.014  05/23/99 to 05/28/99
4.8

Transaction  no data found

Overhead                 0.29

             -----    ------    ------   -----

TOTAL        18373     2.49    3845.0   0.461
```

各カラムについて、以下に説明します。

| カラム | 説明 |
|---|---|
| Type | 記録されているデータの一般的な型です。Overhead という特殊な型のデータがあります。<br>Overhead は、ログファイルによって占められている (または予約されている) ディスクスペース量と走査されたデータレコードによって実際に使用されているスペース量との比率です。<br>ログファイル全体が走査されない場合、Overhead には走査されないデータレコードが含まれます。ファイル全体が走査された場合、Overhead は、ファイル内でデータをブロック化するときの非効率性およびファイルアクセスサポート構造を説明します。抽出ログファイルは、位置決めの迅速化を目的とした追加サポート構造を備えているため、通常は生ログファイルよりもオーバーヘッドが高くなります。 |
| Total | 各データ型について走査された合計レコード数とディスクスペースです。 |
| Each Full Day | scope の実行により 24 時間あたりに使用されるレコードの数とディスクスペースの使用量です。 |
| Dates | 走査された各データ型のデータレコードに関して有効な最初の日付と最後の日付です。 |
| Full Days | このデータ型に関して走査された時間を日数 (24 時間単位) で表した数字です。<br>scope の適用範囲が操作時間の 100 %と一致しない場合、Full Days は開始日から終了日までの日数と等しくなりません。 |

TOTAL 行 (リストされているデータの最下部) により、ディスクスペースの使用量がわかり、毎日蓄積されるデータ量を予想できます。

## ログファイルの空きスペースの要約

この要約は、走査された各ログファイルについて出力されます。たとえば、次のようになります。

```
The Global      file is now  13.9% full with room for 61 more full
days

The Application file is now  15.1% full with room for 56 more full
days

The Process     file is now  23.5% full with room for 32 more full
days

The Device      file is now   1.4% full with room for 2896 more full
days
```

追加データを格納できる空き容量は、次の要素に基づいて算出されます。

- ファイル内の未使用スペースの量
- 走査された値 (24 時間あたりの記録データのメガバイト数)

megabytes-scanned-per-day の値 (1 日あたりの走査メガバイト数) が低く、現実的でない場合、この計算にはデフォルト値が使用されます。

抽出ファイルを走査する場合、すべてのデータ型が同じ抽出ファイルを共用するため、レポートに表示されるのは 1 行だけです。

# utility のコマンド

この章 では、**utility** プログラムのコマンド について説 明します。この章 は、構 文 をまとめた表 と、アルファベット順 にコマンド を説 明しているコマンド リファレンスから構 成 されています。

**utility** のコマンド とパラメータは、大 文 字 と小 文 字 を組 み合 わせて入 力 できます。コマンド 名 の最 初 の 3 文 字 だけは必 ず入 力 します。たとえば、logfile コマンド は **logfile** と入 力 するか、**log** または **LOG** と省 略 して入 力 できます。

これらのコマンド の使 用 例 については、**utility** プログラムのオンライン ヘルプを参 照 してください。

次 のページの表 では、**utility** のコマンド 構 文 とパラメータをまとめています。

**表 4 utility のコマンド :構 文 とパラメータ**

| コマンド | パラメータ |
|---|---|
| analyze | |
| checkdef | **alarmdef** ファイル |
| detail | onoff |
| exite | |
| guide | |
| list | *filename* または * |
| logfile | logfile |
| menu? | |
| parmfile | parmfile |
| quitq | |
| resize | globalapplicationprocessdevicetransactiondays=maxdayssize=max MBempty=daysspace=MByesnomaybe |
| scan | *logfile*<br>(list、start、stop、detail の各コマンド も操 作 に影 響 します) |
| show | all |
| sh! | system command |
| start | *date* [*time*]<br>today [-*days*] [*time*]<br>last [-*days*] [*time*]<br>first [+*days*] [*time*] |
| stop | *date* [*time*]<br>today [-*days*] [*time*]<br>last [-*days*] [*time*]<br>first [+*days*] [*time*] |

# analyze

analyze コマンドは、アラーム定義 (**alarmdef**) ファイル内のアラーム定義と比較してログファイル内のデータを分析し、結果として生じるアラームのステータスと活動状況をレポートするときに使用します。analyze コマンドを発行する前に、checkdef コマンドを実行してアラーム定義構文を確認する必要があります。また、Checkdef は analyze と共に使用するアラーム定義ファイル名を設定し、保存します。analyze の前に checkdef を実行しないと、アラーム定義ファイル名を入力するように促すプロンプトが表示されます。

コマンド ライン モードを使用している場合は、デフォルトのアラーム定義ファイル **/var/opt/perf/alarmdef** が使用されます。

アラーム定義の詳細は、「パフォーマンス アラーム」を参照してください。

**構文**

***analyze***

**使い方**

analyze コマンドを発行すると、**alarmdef** ファイルのアラーム定義と比較して、データソース構成ファイル **datasources** で指定されているログファイルが分析されます。

**analyze** コマンドにより、システム上で収集した履歴データとアラーム定義が適合しているかを評価できます。また、アラーム定義が分析用ワークステーション上で非常に多数のアラームを発生させるか、少数のアラームを発生させるかを決定できます。

さらに、条件が満たされるとPRINT 文により情報を出力できるため、アラーム定義ファイル内の定義 (IF 文) によりデータ分析を実行できます。アラーム定義での IF 文および PRINT 文の使い方については、9 章「パフォーマンス アラーム」を参照してください。

オプションとして、**start**、**stop**、**detail** の各コマンドを **analyze** と共に実行して、分析プロセスをカスタマイズできます。これらのコマンドは次の順序で指定します。

```
checkdef
start
stop
detail
analyze
```

特定期間に収集されたログファイル データを分析する場合は、start コマンドとstop コマンドを使用します(start コマンドとstop コマンドについては、この章で後述します)。

analyze コマンドを実行中に、alarm start、end、repeat status ステータスなどのアラーム イベントと、関連する PRINT 文のテキストがリストされます。また、PRINT 文内のテキストは、IF 文の条件が真になるとリストされます。EXEC 文は実行されませんが、リストされるため、実行内容を参照できます。アラーム要約レポートは、アラームの発生回数と各アラームがアクティブ (オン) であった時間を示します。回数には、alarm starts とrepeats が含まれますが、alarm ends は含まれません。

アラーム要約レポートのみを参照する場合は、detail off コマンドを発行します。ただし、コマンド ライン モードを使用している場合は、detail off がデフォルトであるため、アラームのイベントと要約を参照するには、-D を指定する必要があります。

**例**

checkdef コマンドが、**alarmdef** ファイル内のアラーム定義構文を確認し、後で **analyze** コマンドで使用するために alarmdef ファイルの名前を保存します。start today コマンドは、今日記録されたデータのみの分析を指定します。最後に、analyze コマンドが **alarmdef** ファイルのアラーム定義と比較して、**datasources** ファイルで指定されているデフォルトの SCOPE データソース内のログファイルを分析します。

```
utility>
checkdef /var/opt/perf/alarmdef
start today
analyze
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

**utility -xc -D -b today -xa**

# checkdef

checkdef コマンドは、アラーム定義ファイル内のアラーム定義の構文を確認し、発見された警告またはエラーをレポートするときに使用します。また、このコマンドは analyze コマンドと共に使用するアラーム定義ファイル名を設定し、保存します。

アラーム定義構文とアラーム定義の指定方法については、9 章「パフォーマンス アラーム」を参照してください。

**構文**

**checkdef [/directorypath/alarmdef]**

**パラメータ**

| | |
|---|---|
| alarmdef | 任意のアラーム定義ファイル名です。ユーザー指定のファイル、またはデフォルトの **alarmdef** ファイルを指定します。ディレクトリパスを指定しない場合、現在のディレクトリが検索されます。 |

**使い方**

アラーム定義を正しいと判断した場合は、analyze コマンドを使用してログファイル内のデータと比較して、アラーム定義を処理できます。

バッチ モードで、アラーム定義ファイルを指定しない場合は、デフォルトの alarmdef ファイルが使用されます。

対話型モードでは、アラーム定義ファイルを指定しない場合、ファイルの指定を促すプロンプトが表示されます。

**例**

checkdef コマンドが、**alarmdef** ファイル内のアラーム定義構文を確認し、後で analyze コマンドで使用するために **alarmdef** ファイルの名前を保存します。

```
utility>
checkdef /var/opt/perf/alarmdef
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

**utility -xc**

# detail

detail コマンドは、analyze、parmfile、scan の各レポートに出力する詳細レベルを制御するときに使用します。

デフォルトは、対話型モードおよびバッチ モードでは detail on であり、コマンド ライン モードでは detail off です。

**構文**

| | |
|---|---|
| **detail** | **[on]** |
| | **[off]** |

**パラメータ**

| | |
|---|---|
| on | **parm** ファイルの有効な内容を **parm** ファイルのエラーと共に出力します。analyze と scan の完全なレポートを出力します。 |
| off | **parm** ファイル レポートでは、アプリケーションの定義は出力されません。scan レポートでは、scope の収集時間、初期 **parm** ファイルのグローバル情報、アプリケーションの定義は出力されません。analyze レポートでは、アラーム イベントとアラーム動作は出力されません。 |

**使い方**

analyze、scan、parmfile の各コマンドと共に detail コマンドを使用する方法については、この章の「analyze」、「parmfile」、「scan」のコマンド説明を参照してください。

**例**

detail コマンドの使用例については、この章の「analyze」、「parmfile」、「scan」のコマンド説明を参照してください。

# exit

exit コマンドは、**utility** プログラムを終了するときに使用します。exit コマンドは、**utility** プログラムの quit コマンドに相当します。

**構文**

**exit**

**e**

# guide

guide コマンドは、ガイド コマンド モードに移行するときに使用します。ガイド コマンド インターフェイスは **utility** のさまざまなコマンドについてガイドし、一般的で実行可能ないくつかの作業を行うように促すプロンプトを表示します。

**構文**

**guide**

**使い方**

- **utility** の対話型モードからガイド コマンド モードに移行するには、**guide** と入力して **Return** キーを押します。
- パラメータのデフォルト値を適用するには、**Return** キーを押します。
- ガイド コマンド モードを終了し、対話型モードに戻るには、guide> プロンプトで **q** と入力します。

このコマンドでは、パラメータ設定値を組み合わせる必要がありません。このコマンドは、ほとんどのユーザーに有益な結果をもたらす設定値を選択します。

# help

help コマンドは、**utility** プログラムのオンライン ヘルプにアクセスするときに使用します。

**構文**

**help** [*keyword*]

**使い方**

**utility** のコマンドと作業に関する情報 (ヘルプの情報) を得るには、パラメータを入力します。キーワードを入力すると、別のトピックに移動できます。情報が複数のページにわたる場合は、**Return** を押すと次のページが表示されます。ヘルプ システムを終了し、**utility** プログラムに戻るには、**q** または **quit** と入力します。

また、特定のトピックに関するヘルプも要求できます。たとえば、次のようになります。

**help tasks**

または

**help resize parmfile**

この形式の help コマンドを使用すると、指定のトピックのヘルプ テキストを受信しますが、**utility** のコマンド エントリコンテキストは終了していません。対話型モードでヘルプ サブシステムを指定していないため、次の **utility** コマンドを入力する前に **quit** を入力する必要はありません。

# list

list コマンドは、**utility** のすべてのレポートの出力ファイルを指定するときに使用します。出力されるレポートの内容は、list コマンドの後に発行される別のコマンドによって異なります。たとえば、list コマンドの後に logfile、detail on、scan の各コマンドを使用すると、ログ ファイルの詳細要約レポートのリスト ファイルが作成されます。

**構文**

**list [***filename***]|***

* は stdout に対する出力を設定します。

**使い方**

レポートのリスト ファイルを指定するには、次の 2 つの方法があります。

- 次 のように入 力して、**utility** プログラムを呼 び出 すときに stdout をリダイレクトします。

  **utility > utilrept**

- 次 のように入 力して、utility を実 行 するときに **list** コマンドを使 用します。

  **list utilrept**

どちらの場 合 も、ユーザーの対 話 とエラーが stderr に出 力 され、レポートが指 定 のファイルに出 力 されます。

list コマンドの filename パラメータは、書 き込 みアクセス権 がある有 効 なファイル名 を表 します。既 存 のファイルがある場 合 、既 存 の内 容 の最 後 に新 しい出 力 が追 加 されます。ファイルがない場 合 は作 成 されます。

現 在 の出 力 ファイルを特 定 するには、パラメータを使 用 せずに list コマンドを発 行 します。

出 力 ファイルが stdout でない場 合 、ほとんどのコマンドは入 力 されたときに出 力 ファイルにエコーされます。

**例**

list コマンドは、抽 出 ログファイル rxlog に関 する要 約 レポートを作 成 します。list utilrept コマンドはディスクファイルに走 査 レポート リストを送 信 します。detail off はレポートの完 全 な詳 細 の指 定 を無 効 にします。scan コマンドは rxlog を読 み取 り、レポートを作 成 します。

list * コマンドはリスト デバイスをデフォルトの stdout に設 定 します。!lp utilrept はシステム プリンタにディスクファイルを送 信 します。

```
utility>

logfile rxlog

list utilrept

detail off

scan

list *

!lp utilrept
```

コマンド ライン引 数 を使 用 して上 記 の操 作 を実 行 するには、次 のように入 力 します。

```
utility -l rxlog -f utilrept -d -xs print utilrept
```

# logfile

logfile コマンドは、ログファイルをオープンするときに使 用 します。**utility** プログラムの関 数 を使 用 するときは、多 くの場 合 ログファイルをオープンする必 要 があります。これを明 示 的 に行 うには logfile コマンドを発 行 し、暗 黙 的 に行 うには別 のコマンドを発 行 します。バッチ モードまたはコマンド ライン モードにあり、ログファイル名 を指 定 しない場 合 は、デフォルトの **/var/opt/perf/datafiles/logglob** ファイルが使 用 されます。対 話 型 モードでログファイル名 を指 定 しない場 合 は、ファイル名 を指 定 するか、デフォルトの **/var/opt/perf/datafiles/logglob** ファイルを適 用 するように促 すプロンプトが表 示 されます。

### 構文

**logfile [*logfile*]**

### 使い方

生ログファイル名または抽出ログファイル名を指定できます。抽出ログファイル名を指定する場合、すべての情報がこのファイルから得られます。グローバルログファイル `logglob` 以外の生ログファイルを指定する必要はありません。**logglob** をオープンすると、別のログファイルに含まれるすべてのデータにアクセスできます。

生ログファイル名は次のとおりです。

| | |
|---|---|
| **logglob** | グローバルログファイル |
| **logappl** | アプリケーションログファイル |
| **logproc** | プロセスログファイル |
| **logdev** | デバイスログファイル |
| **logtran** | トランザクションログファイル |
| **logls** | 論理システムログファイル |
| **logindx** | 索引ログファイル |

ログファイルが正しくオープンされると、ログファイル (1 つまたは複数 ) の内容を示すレポートが印刷または表示されます。たとえば、次のようになります。

Global file:/var/opt/perf/datafiles/logglob version DApplication file:/var/opt/perf/datafiles/logapplProcess file:/var/opt/perf/datafiles/logprocDevice file:/var/opt/perf/datafiles/logdevTransaction file:/var/opt/perf/datafiles/logtranIndex file:/var/opt/perf/datafiles/logindxSystem ID:homerSystem Type 9000/715 S/N 6667778899 O/S HP-UX B.10.20. AData Collector:SCOPE/UX C.02.30File Created:06/14/99Data Covers:27 days to 7/10/99Shift is:All DayData records available are:Global Application Process Disk Volume TransactionMaximum file sizes:Global=10.0 Application=10.0 Process=20.0 Device=10.0 Transaction=10.0 MB

```
The first GLOBAL record is on 06/14/99 at 12:00 AMThe first APPLICATION
record is on 06/25/99 at 12:00 AMThe first PROCESS record is on 07/06/99
at 12:01 AMThe first DEVICE record is on 05/01/99 at 11:50 AMThe first
TRANSACTION record is on 05/01/99 at 11:55 AM The default starting date &
time = 05/01/99 11:50 AM (FIRST + 0)The default stopping date & time =
07/10/99 11:59 PM (LAST - 0)
```

`show` コマンドでオープンしたログ ファイルは、後述のように検証できます。

再度 logfile コマンドを入力すると、随時別のログファイルをオープンできます。現在オープンしているログファイルは、新しいログファイルがオープンされる前にクローズされます。

resize コマンドとscan コマンドを実行するには、ログファイルをオープンする必要があります。現在オープンしているログファイルがない場合は、暗黙的に logfile コマンドが実行されます。

**メモ :** 生ログファイル名は変更しないようにしてください。生ログファイルへのアクセスは、標準ログファイル名が有効であると見なされて行われます。

**メモ :** 複数の生ログファイルセットを同じシステムに配置する必要がある場合、各ファイルセットに別々のディレクトリを作成してください。ログファイル名は変更できませんが、別々のディレクトリは使用できます。任意の方法でログファイルのサイズを変更する場合、すべてのログファイルに対する読み取りアクセスおよび書き込みアクセスが必要です。

# menu

menu コマンドは、使用可能な **utility** のコマンドのリストを出力するときに使用します。

**構文**

**menu**

**例**

```
utility> menu

Command  Parameters       Function

HELP     topic]           Get information on commands and options

GUIDE                     Enter guided commands mode for novice users

LOGFILE  [logname]        Specify a log file to be processed

LIST     [filename|*]     Specify the listing file

START    [startdate time] Set starting date & time for SCAN or ANALYZE

STOP     [stopdate  time] Set ending   date & time for SCAN or ANALYZE

DETAIL   [ON|OFF]         Set report detail for SCAN, PARMFILE, or
ANALYZE

SHOW     [ALL]            Show the current program settings

PARMFILE [parmfile]       Check parsing of a parameter file

SCAN     [logname]        Read the log file and produce a summary
report

RESIZE   [GLOB|APPL|PROC|DEV|TRAN]
         [DAYS=][EMPTY=]  Resize raw log files

CHECKDEF [alarmdef]       Check parsing and set the alarmdef file

ANALYZE                   Analyze the log file using the alarmdef file

! or Sh  [command]        Execute a system command

MENU or ?                 List the commands menu (This listing)

EXIT or Q                 Terminate the program
```

```
utility>
```

# parmfile

parmfile コマンドは、データ収集に使用する Performance Collection Component の parm ファイルの設定値を表示し、構文を確認するときに使用します。

**構文**

**parmfile [***/directorypath/parmfile***]**

**使い方**

次のいずれかを行うときに、parmfile コマンドを使用します。

- **parm** ファイルの構文の警告に関する調査と、結果として生じる設定値のレビュー。すべてのパラメータが正しい構文であることが確認され、エラーがレポートされます。構文の確認が完了すると、適用可能な設定値だけがレポートされます
- アプリケーション定義用の空き容量の検索
- detail on を指定したときの **parm** ファイルの有効な内容、無効にならないデフォルト設定値、アプリケーション定義の出力

バッチ モードでは、**parm** ファイル名を指定しないと、デフォルトの **parm** ファイルが使用されます。

対話型モードでは、**parm** ファイル名を指定しないと、指定するように促すプロンプトが表示されます。

**例**

parmfile コマンドは、現在の **parm** ファイルの構文を確認し、警告やエラーをレポートします。**detail on** は記録しているパラメータの設定をリストします。

```
utility>
detail on
parmfile parm
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
utility -xp -D
```

# quit

quit コマンドは、**utility** プログラムを終了するときに使用します。quit コマンドは、**utility** プログラムの exit コマンドに相当します。

**構文**

```
quit
```

```
q
```

# resize

resize コマンドは、使用している生ログファイルセットのスペースを管理するときに使用します。ファイルとファイル内部の制御構造との調整を保存するために生ログファイルのサイズを変更するときは、この

プログラムを使用する必要があります。別のツールを使用する場合、これらの制御構造の有効性を削除するか、破棄してください。

**utility** プログラムは、抽出ファイルのサイズを変更するときには使用できません。抽出ファイルのサイズを変更する場合は、**extract** プログラムを使用し、新しい抽出ログファイルを作成します。

**構文**

| | | | | |
|---|---|---|---|---|
| resize | [global ]<br>[application]<br>[process ]<br>[device ]<br>[transaction] | [days=maxdays]<br>[size=maxMB ] | [empty=days]<br>[space=MB ] | [yes ]<br>[no ]<br>[maybe] |

**パラメータ**

| | |
|---|---|
| `log file type` | サイズを変更する生データ型 (グローバル、アプリケーション、プロセス、デバイス、トランザクションのいずれか) を指定します。生データ型はそれぞれ、生ログファイルの **logglob**、**logappl**、**logproc**、**logdev**、**logtran** に対応しています。データ型を指定せずに、バッチモードで **utility** を実行している場合は、バッチジョブが終了します。対話型モードで **utility** を実行している場合は、現在存在するログファイルに基づいてデータ型を指定するように促すプロンプトが表示されます。 |
| days & size | ログファイルの最大サイズを指定します。実際のサイズは、ファイル内のデータの量によって異なります。 |
| empty & space | サイズ変更操作の完了後に、ファイルに必要な最小空き容量を指定します。この値は、ログファイル内に現在存在するデータをサイズ変更プロセスで削除する必要があるかを決定するときに使用します。 |

サイズ変更の操作後に、ログファイルは指定した日数までいっぱいにならないと予測できます。サイズ変更はログファイルがいっぱいになると行われるため、resize コマンドのこの機能を使用すると、scope コレクタによるログファイルのサイズ変更の回数を最小にできます。resize を使用して、ログファイル内に強制的に一定の空き容量を確保すると、必要なときにログファイルのサイズを変更できます。

days および empty には日数で値を入力し、size および space にはメガバイト単位で値を入力します。日数は、ログファイルの 1 日あたりの平均メガバイト数を使用して、メガバイト単位に変換されます。この変換要素は、記録するデータ型やシステムの特定の特徴によって異なります。

resize コマンドを発行する前に、既存のログファイルに対して scan コマンドを発行すると、より正確な 1 日あたりの平均メガバイト数変換要素が得られます。scan は、システムの累計率を測定します。scan を実行しない場合、または測定した変換要素が妥当でない場合、resize コマンドは各データ型のデフォルトの変換要素を使用します。

| | |
|---|---|
| `yes` | サイズ変更を無条件で実行することを指定します。**utility** を対話型モード以外で |

| | |
|---|---|
| | 実行している場合は、これがデフォルトの動作になります。対話型モードで **utility** を実行し、動作を指定しない場合は、動作の指定を促すプロンプトが表示されます。 |
| no | サイズ変更を実行しないことを指定します。このパラメータは、サイズ変更レポートを表示し、そのときにサイズ変更を実行しない場合の動作として指定できます。 |
| maybe | ファイルのサイズ変更を実行するか実行しないかを **utility** により決定することを指定します。このパラメータは、ログファイル内の現在の空き容量 (サイズ変更前) と、resize コマンドで指定した容量に基づいて utility がこの決定を行うように強制します。現在のログファイルの空き容量が少なくとも指定した容量と同じだけある場合、サイズ変更は*行われません*。現在のログファイルの空き容量が指定した容量より小さい場合は、サイズ変更が行われます。 |
| maybe (*続き*) | ログファイルからデータを削除せずにサイズ変更が可能な場合 (既存のデータを削除せずに最大ログファイルサイズを増減する場合など)、サイズ変更が行われます。<br>maybe パラメータは、主に定期的なバッチ操作で使用するために用意されています。このように resize コマンドを使用する方法については、後述の「例」を参照してください。 |

デフォルトのサイズ変更パラメータを、次の表に示します。

**表 5 デフォルトのサイズ変更パラメータ**

| パラメータ | 対話型モードで実行した場合 | バッチで実行した場合 |
|---|---|---|
| log file type | 有効なログファイル型を入力するように促すプロンプトが表示されます。 | デフォルトなし。これは必須パラメータです。 |
| dayssize | 現在のファイルサイズ。 | 現在のファイルサイズ。 |
| 空き容量 | 現在の空き容量、または現在ファイルにあるすべてのデータを保存するために必要な空き容量のうちどちらか小さい方。 | 現在の空き容量、または現在ファイルにあるすべてのデータを保存するために必要な空き容量のうちどちらか小さい方。 |
| yesnomaybe | レポートされたディスクスペース結果に従うことを促すプロンプトが表示されます。 | Yes で、サイズ変更が行われます。 |

### 使い方

ログファイルのサイズを変更する前に、2 章の「データ収集の終了と再開始」に記載されている手順で、Performance Collection Component を終了する必要があります。

生ログファイルはサイズ変更を行う前にオープンする必要があります。resize コマンドを発行する前に、logfile コマンドで生ログファイルをオープンします。ファイルは、別のプロセスではオープンできません。

resize コマンドは、元のログファイルを削除する前に、TMPDIR 環境変数で設定されているディレクトリに新しいファイル scopelog を作成します。環境変数 TMPDIR が設定されていない場合、一時的

な場所として /var/tmp ディレクトリ (IBM AIX 4.1 以降の場合、/tmp) が使用されます。サイズ変更の手順を行う前に、元のログファイルを保持するために、TMPDIR で指定したディレクトリまたは /var/tmp ディレクトリ (IBM AIX 4.1 以降の場合、/tmp ディレクトリ) に十分なディスクスペースがあることを確認してください。

サイズ変更後は、ログファイルはデータと空きスペースで構成されます。保存されたデータは、最大ファイルサイズから必要な空きスペースを引いた大きさとして計算されます。サイズ変更操作中に削除されたデータは失われます。ログファイルデータを長期間保存するには、extract を使用して、サイズ変更操作を行う前にこのデータを抽出ファイルにコピーします。

## resize コマンドのレポート

生ログファイルのサイズを変更すると、標準レポートが作成されます。このレポートは、相互に関係する 3 つのディスクスペースカテゴリ (最大ファイルサイズ、データレコード、空きスペース) について、サイズ変更の前後の値を示します。たとえば、次のようになります。

```
resize global days=120;empty=10

empty space raised to match file size and data records

final resizing parameters:

file: logglob                      megabytes / day: 0.101199

                ---currently-----    --after resizing---

maximum size:   65 days (  6.6 mb)   120 days ( 12.1 mb)  83% increase

data records:   61 days (  6.2 mb)    61 days (  6.2 mb)  no data
removed

empty space:     4 days (  0.5 mb)    59 days (  6.0 mb)  1225%
increase
```

1 日あたりのメガバイト数は、日数とメガバイトの変換に使用されます。これは、走査関数で得られる値またはサイズ変更するデータ型のデフォルトです。

右端のカラムは、ログファイルスペースの各カテゴリについてのネット変更の要約です。最大サイズと空きスペースは増加や削減が可能で、変更されないこともあります。サイズ変更中に、データレコードのデータはいずれのデータも削除されない場合、または指定した量のデータが削除される場合があります。

対話形式でサイズ変更が行われ、1 つ以上のパラメータがデフォルトである場合、予備のサイズ変更レポートを作成できます。このレポートは、現在のログファイルの内容と指定されたパラメータを要約し、未指定のパラメータに関する問題を解決するために提示されます。たとえば、次のようになります。

```
resize global days=20

file resizing parameters (based on average daily

space estimates and user resizing parameters)

file: logglob                        megabytes / day: 0.101199

                -----currently----    --after resizing--

maximum size:   65 days (  6.6 mb)    20 days (  2.0 mb)

data records:   61 days (  6.2 mb)    ??
```

```
empty space:     4 days (  0.5 mb)    ??
```

この例では、最終的なサイズ変更レポートを作成する前に、ファイルの空き容量を指定するように促すプロンプトが表示されます。対話形式のサイズ変更に対して動作パラメータを指定しない場合、最終的なサイズ変更レポートに続いて、ログファイルのサイズ変更を指定するように促すプロンプトが表示されます。

## 例

次のコマンドは、生プロセスログファイルのサイズを変更するときに使用します。日数の計算の精度を向上させるため、サイズ変更の前に走査を実行します。

```
logfile /var/opt/perf/datafiles/logglob
detail off
scan
resize process days=60 empty=30 yes
```

days=60 は、最大 60 日分のデータを保持することを指定します。empty=30 はこのファイルの 30 日分のスペースが現在空きスペースであることを指定します。つまり、合計 60 日分のスペースに 30 日分のデータがある状態で、あと30 日分の空きスペースを残すために、ファイルのサイズ変更が行われます。yes は、現在の空きスペースにかかわらず、サイズ変更を行うことを指定します。

次の例は、バッチモードで resize コマンドを使用して、ログファイルがこれから 1 週間はいっぱいにならないようにする方法を示します (強制的に scope によりファイルのサイズを変更します)。at コマンドで 7 日分、または多めに 10 日分などと最小容量を指定して、cron スクリプトをスケジュールできます。

次のシェルスクリプトは、以下の操作を行います。

```
echo detail off                          > utilin

echo scan                                >> utilin

echo resize global       empty=10 maybe >> utilin

echo resize application  empty=10 maybe >> utilin

echo resize process      empty=10 maybe >> utilin

echo resize device       empty=10 maybe >> utilin

echo quit                                >> utilin

utility < utilin > utilout 2> utilerr
```

10 日分以上の空きスペースが現在ログファイルに存在する場合、yes の代わりに maybe を指定すると、サイズ変更操作が行われません。デフォルトでは、最大ファイルサイズが各ファイルの現在の最大ファイルサイズになることに注意してください。この指定により、このスクリプトに影響を与えずにファイルのサイズを新しい最大サイズに変更できます。

# scan

scan コマンドは、ログファイルを読み取り、その内容に関するレポートを作成するときに使用します(レポートの詳細は、3 章の「utility 走査レポートの詳細」を参照してください)。

### 構文

**scan**

### 使い方

scan コマンドを実行するには、ログファイルをオープンする必要があります。走査するログファイルは、次のファイルのうち最初に指定されたものです。

- scan コマンドで指定したログファイル
- 前のコマンドで最後にオープンしたログファイル
- デフォルトのログファイル

この場合、対話型モードでは、必要に応じてデフォルト ログファイル名を無効にするように促すプロンプトが表示されます。

次のコマンドが走査関数の操作に影響します。

| | |
|---|---|
| `detail` | レポートの詳細レベルを指定します。デフォルトの detail on は完全な詳細を指定します。 |
| `list` | 出力を別のファイルにリダイレクトします。デフォルトでは、標準のリスト デバイスにリストします。 |
| `start` | 走査する最初のログファイルレコードの日時を指定します。デフォルトは、ログファイルの開始日時です。 |
| `stop` | 走査する最後のログファイルレコードの日時を指定します。デフォルトは、ログファイルの終了日時です。 |

detail、list、start、stop の各コマンドの詳細は、この章のそれぞれの説明を参照してください。

scan コマンドのレポートは 12 のセクションから構成されます。scan を発行する前に detail コマンドを発行すると、レポートに入れるセクションを制御できます。

次の 4 つのセクションは、(detail off を指定している場合も) 必ず出力されます。

- 実際の走査の開始日時および終了日時
- コレクタの適用範囲の要約
- ログファイルの内容の要約
- ログファイルの空きスペースの要約

detail on (デフォルト) を指定している場合は、次のセクションが出力されます。

- 初期 **parm** ファイルのグローバル情報およびシステム構成情報
- 初期 **parm** ファイルのアプリケーション定義
- **parm** ファイルのグローバル変更内容
- parm ファイルのアプリケーション追加通知および削除通知
- コレクタのオフタイム通知
- アプリケーション固有の要約レポート

アプリケーション データが走査された場合は、(detail off を指定している場合も) 次のセクションが必ず出力されます。

- アプリケーションの全体的な要約

プロセス データが走査された場合は、(detail off を指定している場合も) 次のセクションが必ず出力されます。

- プロセスの記録理由の要約

**例**

現在のデフォルト グローバル ログ ファイルの走査を開始します。対象は、1999 年 6 月 1 日 7:00 AM から現在の日時までに記録されたレコードです。

```
utility>
logfile /var/opt/perf/datafiles/logglob
detail on
start 6/1/99 7:00 am
scan
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
utility -D -b 6/1/99 7:00 am -xs
```

# sh

sh は、**utility** を終了せずにシェル コマンドを入力するときに使用します。**sh** またはエクスクラメーション マーク (**!**) の後にシェル コマンドを指定します。

**構文**

**sh** または **![shell command]**

**パラメータ**

| | |
|---|---|
| `sh ls` | ls コマンドを実行し、**utility** に戻ります。 |
| `!ls` | 上記と同じです。 |

**使い方**

1 つのコマンドを実行すると、自動的に **utility** に戻ります。複数のシェル コマンドを発行するときに、1 つのコマンドを実行するたびに **utility** に戻らないようにする場合は、次のように新しいシェルを開始します。たとえば、次のようになります。

```
sh ksh
```

or

```
!ksh
```

# show

show コマンドは、オープンしたファイルの名前と設定可能な **utility** パラメータのステータスのリストを作成するときに使用します。

**構文**

```
show [all]
```

**例**

show を使用すると、次のようなリストが作成されます。

```
Logfile:/var/opt/perf/datafiles/logglob List:     "stdout"Detail:ON
for ANALYZE, PARMFILE and SCAN functions


The default starting date & time = 10/08/99 08:17 AM (FIRST + 0)The
default stopping date & time = 11/20/99 11:59 PM (LAST  - 0)The
default shift = 12:00 AM - 12:00 AM
```

**メモ :** デフォルトのシフト時間が情報として示されます。utility ではシフト時間は変更できません。

show all を使用すると、次の例のようにより詳細なリストが作成されます。

**ログファイル:** `/var/opt/perf/datafiles/logglob`

```
    Global      file:/var/opt/perf/datafiles/logglob
    Application file:/var/opt/perf/datafiles/logappl
    Process     file:/var/opt/perf/datafiles/logproc
    Device      file:/var/opt/perf/datafiles/logdev
    Transaction file:/var/opt/perf/datafiles/logtran
    Index       file:/var/opt/perf/datafiles/logindx
    System ID:homer
    System Type 9000/715 S/N 66677789   OS/ HP-UX B.10.20 A
    Data Collector:SCOPE/UX C.02.30
    File created:10/08/99
    Data Covers:44 days to 11/20/99
    Shift is:All Day
Data records available are:
  Global Application Process Disk Volume Transaction

Maximum file sizes:
  Global=10.0 Application=10.0 Process=20.0 Device=10.0 Transaction
10.0 MB

List    "stdout"
Detail  ON for ANALYZE, PARMFILE and SCAN functions

The default starting date & time = 10/08/99 11:50 AM (FIRST + 0)
The default stopping date & time = 11/20/99 11:59 PM(LAST - 0)
The default shift = 12:00 AM - 12:00 AM
```

# start

start コマンドは、走査または分析するログファイルのサブセットの開始日時を指定するときに使用します。start により、特定の日時に記録されたデータから scan プロセスまたは analyze プロセスを開始します。

デフォルトの開始日時は、logfile コマンドで現在オープンしているログファイルに含まれる最初のレコード (任意の型 ) の日時に設定されています。

**構文**

| | | | |
|---|---|---|---|
| start | *[date* **[today** **[last** **[first** | *[time]]* *[-days]* *[-days* *][+days]* | *[time]]* *[time]]* *[time]]* |

**パラメータ**

| | |
|---|---|
| `date` | 日付フォーマットは、使用しているシステムに構成されている母国語によって異なります。母国語を使用しない場合やデフォルトの言語を C に設定している場合、日付フォーマットは *mm/dd/yy* (月 / 日 / 年 ) になります。たとえば、1999 年 6 月 30 日は 06/30/99 になります。 |
| `time` | 時刻フォーマットも、使用している母国語によって異なります。C の場合、フォーマットは *hh:mm AM* または *hh:mm PM* (12 時間制の時 : 分の後に AM または PM を付ける形式 ) になります。たとえば、午前 7 時は 07:00 AM になります。24 時間制の時刻もすべての言語で使用できます。たとえば、11:30 PM を 23:30 とすることもできます。<br>日付または時刻を不適切なフォーマットで入力すると、正しいフォーマットの例が表示されます。<br>開始時刻を指定しないと、午前 0 時 (12:00 AM) と見なされます。指定日の午前 0 時が開始時刻の場合は、その日の始まり (24 時間制の場合 00:00) に開始されます。 |
| `today` | 現在の日付を指定します。today-*days* パラメータは、現在の日付からさかのぼる日数を指定します。たとえば、today-1 は前日を示し、today-2 は前前日を示します。 |
| `last` | ログファイルに含まれる最終日を表すときに使用します。last-*days* パラメータは、ログファイル内の最終日からさかのぼる日数を指定します。 |
| `first` | ログファイルに含まれる最初の日を表すときに使用します。first+*days* パラメータは、ログファイル内の最初の日から経過した日数を指定します。 |

**使い方**

start コマンドは、非常に大きなログファイルに対して、ファイル全体の走査や分析を行わないときに役立ちます。また、このコマンドを使用して走査するデータの期間を指定できます。たとえば、**today-14** と指定すると、現在の日付の 14 日前から記録されたデータのログファイルを走査できます。

stop コマンドを使用することで、走査するログファイルレコードをさらに制限できます。

母国語のサポートがシステムにインストールされているかわからない場合は、**utility** を実行する前に次の文を発行して、次のように **utility** が C の日時フォーマットを使用するように指定できます。

```
LANG=C; export LANG
```

C シェルでは、次の文を実行します。

```
setenv LANG C
```

**例**

デフォルト グローバル ログファイルの走査を開始します。対象は、1999 年 8 月 5 日 8:00 AM から現在の日時までに記録されたレコードです。

```
utility>
logfile /var/opt/perf/datafiles/logglob
detail on
start 8/5/99 8:00 AM
scan
```

コマンド ライン引 数を使 用して上 記 の操 作を実 行 するには、次 のように入 力します。

**utility -D -b 8/5/99 8:00 AM-xs**

# stop

stop コマンド は、走 査 または分 析 するログ ファイルのサブセット の終 了 日 時を指 定 するときに使 用します。stop により、特 定 の日 時 に記 録されたデータで走 査 プロセスまたは分 析 プロセスを終 了 できます。

デフォルト の終 了 日 時 は、logfile コマンド で現 在 オープンしているログ ファイルに含 まれる最 後 のレコード (任 意 の型 ) の日 時 に設 定 されています。

**構 文**

| | | | |
|---|---|---|---|
| **stop** | *[date*<br>**[today**<br>**[last**<br>**[first** | *[time]]*<br>*[-days]*<br>*[-days]*<br>*[+days]* | <br>*[time]]*<br>*[time]]*<br>*[time]]* |

**パラメータ**

| | |
|---|---|
| date | 日 付 フォーマット は、使 用しているシステムに構 成されている母 国 語 によって異 なります。母 国 語 を使 用しない場 合 やデフォルト の言 語 を C に設 定している場 合 、日 付 フォーマット は *mm/dd/yy* (月 / 日 / 年 ) になります。たとえば、1999 年 6 月 30 日 は 06/30/99 になります。 |
| time | 時 刻 フォーマット も、使 用している母 国 語 によって異 なります。C の場 合 、フォーマット は *hh:mm AM* または *hh:mm PM* (12 時 間 制 の時 : 分 の後 に AM または PM を付 ける形 式 ) になります。たとえば、午 前 7 時 は 07:00 AM になります。24 時 間 制 の時 刻 もすべての言 語 で使 用 できます。たとえば、11:30 PM を 23:30 とすることもできます。<br>日 付 または時 刻 を不 適 切 なフォーマット で入 力 すると、正 しいフォーマット の例 が表 示 されます。<br>終 了 時 刻 を指 定 しないと、午 前 0 時 の 1 分 前 (11:59 PM) と見 なされます。指 定 日 の午 前 0 時 (12:00 AM) が終 了 時 刻 の場 合 、その前 日 の終 わり (24 時 間 制 の場 合 23:59) に終 了 します。 |
| today | 現 在 の日 付 を指 定 します。today-*days* パラメータは、現 在 の日 付 からさかのぼる日 数 を指 定 します。たとえば、today-1 は前 日 を示 し、today-2 は前 前 日 を示 します。 |
| last | ログ ファイルに含 まれる最 終 日 を表 すときに使 用 します。last-*days* パラメータは、ログ ファイル内 の最 終 日 からさかのぼる日 数 を指 定 します。 |
| first | ログ ファイルに含 まれる最 初 の日 を表 すときに使 用 します。first+*days* パラメータは、ロ |

| | |
|---|---|
| | グファイル内の最初の日から経過した日数を指定します。 |

**使い方**

stop コマンドは、非常に大きなログファイルに対して、ファイル全体の走査を行わないときに役立ちます。また、このコマンドを使用して走査するデータの期間を指定できます。たとえば、現在の日付の1か月前から記録された7日分のデータのログファイルを走査できます。

母国語のサポートがシステムにインストールされているかわからない場合は、**utility** を実行する前に次の文を発行して、次のように **utility** が C の日時フォーマットを使用するように指定できます。

```
LANG=C; export LANG
```

C シェルでは、次の文を実行します。

```
setenv LANG C
```

**例**

1999 年 7 月 5 日 8:00 AM から記録されたレコードについて、14 日分のデータの走査を開始し、1999 年 7 月 18 日 11:59 PM に記録された最後のレコードで終了します。

```
utility>
logfile /var/opt/perf/datafiles/logglob
detail on
start 7/5/99 8:00 am
stop 7/18/99 11:59 pm
scan
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

**utility -D -b 7/5/99 8:00 am -e 7/18/99 11:59pm -xs**

# extract プログラムの使い方

extract プログラムには 2 つの主 要 な関 数 が含 まれており、生 ログ ファイルからデータを抽 出して、抽 出 ログ ファイルに書 き込 むことができます。extract では、表 計 算 ソフトなどの分 析 用 製 品 で使 用 できるように、ログ ファイル データをエクスポートすることもできます。

**注:** 初 めて HP Operations エージェントをインストールした場 合 は、extract を機 能 させるには、ファイルのインストールを完 了 するために各 サービスを起 動 する必 要 があります。

extract 関 数 および export 関 数 は、ログ ファイルからデータをコピーします。すなわち、データは削 除 されません。

Performance Collection Component では、次 の 3 種 類 のログ ファイルを使 用 します。

- scope ログ ファイル ー scope コレクタにより Performance Collection Component 内 で収 集 されるデータが含 まれます。
- 抽 出 ログ ファイル ー scope 生 ログ ファイルから抽 出 されるデータが含 まれます。
- データ ソース統 合 (DSI:data source integration) ログ ファイル ー アプリケーションやデータベースなどの外 部 ソースにより収 集 されるユーザー定 義 のデータが含 まれます。このデータは、後 で Performance Collection Component の DSI プログラムにより記 録 されます。

extract プログラムを使 用 すると、次 の作 業 を行 うことができます。

- scope 生 ログ ファイルからデータのサブセットを抽 出 し、抽 出 ログ ファイル フォーマットにします。抽 出 ログ ファイル フォーマットは、アーカイブ、システム間 の転 送 、HP Performance Manager での分 析 に適 したフォーマット です。データは、DSI ログ ファイルからは抽 出 できません。
  extract プログラムにロールバックする場 合
- 抽 出 フォーマット ファイルからデータを抽 出 またはエクスポートし、そのデータを既 存 の抽 出 ログ ファイルに追 加 し、データ型 や日 付 、シフト (時 刻 ) によって整 理 することで、アーカイブ ログ ファイル データを管 理 します。
- 生 scope ログ ファイルまたは抽 出 scope ログ ファイルおよび DSI ログ ファイルからデータをエクスポートし、レポート や分 析 、表 計 算 ソフト や同 様 の分 析 ソフト へのインポートに適 した ASCII、binary、datafile、WK1 (表 計 算 シート ) のいずれかのフォーマットにします。

**注:**

1. extract 関 数 では要 約 データは作 成 できません。要 約 データは export 関 数 でのみ作 成 することができます。
2. 抽 出 関 数 またはエクスポート関 数 は以 下 のように制 限 されます。
   - 抽 出 関 数 からの出 力 ファイルは、最 大 3.5 ギガバイト です。
   - エクスポート関 数 からの出 力 ファイルは、最 大 4 ギガバイト です。

実 行 される多 様 な作 業 例 と extract コマンド の使 用 例 については、extract プログラムのオンライン ヘルプを参 照 してください。

この章 では次 の内 容 について説 明 します。

extract プログラムの実 行

インタラクティブ モードの使用

extract のコマンド ライン インターフェイス

エクスポート関数の概要

# extract プログラムの実行

extract プログラムを実行するには、次の3つの方法があります。

## 構文

コマンド ライン インターフェイスの構文は、その他のプログラムで使用される標準的な UNIX コマンド ライン インターフェイスと同様であり、この章で詳細を説明します。

対話型モードとバッチ モードのコマンド構文は同一です。つまり、コマンドの後に1つ以上のパラメータを指定します。コマンドは、どのような順序で入力してもかまいません。コマンドに関連付けるパラメータがある場合、そのパラメータは対応するコマンドの直後に入力する必要があります。

パラメータには、必須のパラメータ (デフォルトなし) とオプションのパラメータ (デフォルトあり) の2種類があります。extract プログラムがこれらのパラメータを扱う方法は、extract プログラムの実行モードによって異なります。

対話型モードでオプションのパラメータが欠落している場合は、プログラムがデフォルトのパラメータを表示します。ユーザーはその引数を確定するか、無効にする必要があります。必須のパラメータが欠落している場合、プログラムがパラメータを入力するように促すプロンプトを表示します。

バッチ モードでオプションのパラメータが欠落している場合、プログラムはデフォルト値を使用します。必須のパラメータが欠落している場合、プログラムは終了します。

対話型モードの場合とコマンド ラインおよびバッチ モードの場合では、エラーと欠落データが異なる方法で処理されます。対話型モードではデータの追加や間違いの訂正ができますが、コマンド ラインやバッチ モードではできません。

# 対話型モードの使い方

extract プログラムの対話型モードを使用するには、特定の作業を行うための一連のコマンドを発行する必要があります。

たとえば、2003 年の 5 月 15 日から収集されたアプリケーション データをデフォルトのグローバル ログ ファイルからエクスポートする場合、extract プログラムを呼び出した後に次のコマンドを発行します。

```
logfile /var/opt/perf/datafiles/logglob

application detail

start 5/15/2003

export
```

logfile コマンドがデフォルトのグローバル ログ ファイル /var/opt/perf/datafiles/logglob をオープンします。start コマンドが 2003 年 5 月 15 日以降に記録されたデータだけをエクスポートすることを指定します。export コマンドがデータのエクスポートを開始します。

# extract のコマンド ライン インターフェイス

対話型モードとバッチ モードのコマンド構文のほかに、コマンド ライン インターフェイスを使用してもコマンド オプションと引数を extract プログラムに渡すことができます。extract プログラムをシェル スクリプトで簡単に呼び出せるようにし、UNIX のパイプへの入出力のリダイレクトを可能にすることで、コマンド ライン インターフェイスは一般的な UNIX 環境に適合します。

たとえば、コマンド ラインを使用して、前述の「対話型モードの使い方」に示されている例と同様の処理をするには、次のように入力します。

```
extract -l -a -b 5/15/02 -xp
```

コマンド ライン モードでは、グローバル ログ ファイル /var/opt/perf/datafiles/logglob がデフォルトです。このファイルを指定する必要はありません。

次の表で、コマンド ラインのオプションと引数を示します。それぞれのコマンドの説明は、7 章「extract のコマンド」に記載されています。

**表 6:コマンド ラインの引数**

| コマンド オプション | 引数 | | 説明 |
|---|---|---|---|
| -b | date | time | extract 関数や export 関数の開始日時を指定します(7 章の start コマンドを参照)。 |
| -B | | UNIX*start time* | extract 関数や export 関数の UNIX フォーマットでの開始時刻を指定します。 |
| -e | date | time | extract 関数や export 関数の終了日時を指定します(7 章の stop コマンドを参照)。 |
| -E | | UNIX*stop time* | extract 関数や export 関数の UNIX フォーマットでの終了時刻を指定します。 |
| -s | time-time | noweekends | 週末を除く特定期間の開始時刻と終了時刻を指定します(7 章の shift コマンドを参照)。 |
| -l | logfile | | 入力ログファイルを指定します(7 章の logfile コマンドを参照)。デフォルトは /var/opt/perf/datafiles/logglob です。 |
| -r | export template file | | export 関数で使用するエクスポート テンプレート ファイルを指定します(7 章の report コマンドを参照)。 |
| -C | classname | opt | extract または export を実行する scope データ、または export を実行するする自己記述 (DSI) データを指定します(7 章の class コマンドを参照)。<br>opt = detail (デフォルト) summary both off |

| | | | |
|---|---|---|---|
| -i | | | 論理システムから extract または export を実行する scope データを指定します。 |
| -k | | | 抹消されたプロセスのみを抽出します。このオプションを使用する場合は、reptfile に PROC_INTEREST メトリックを加えます。 |
| gapkcdzntuy<br>iGADZNTUYI | | | 次のように、extract または export を実行するデータ型を指定します。<br><br>g = グローバル詳細 (7 章の global コマンドを参照)。デフォルトではグローバル詳細はオフです。<br><br>a = アプリケーション詳細 (7 章の application コマンドを参照)。<br><br>p = プロセス詳細 (7 章の process コマンドを参照)。<br><br>k = プロセス抹消 (7 章の process コマンドを参照)。<br><br>c = 構成詳細 (7 章の configuration コマンドを参照)。<br><br>d = ディスク デバイス詳細 (7 章 disk コマンドの項を参照)。<br><br>z = lvolume 詳細 (7 章の lvolume コマンドを参照)。<br><br>n = netif 詳細 (7 章の netif コマンドを参照)。 |
| gapkcdzntuy<br>iGADZNTUYI<br>(続き) | | | t= トランザクション詳細<br><br>u = CPU 詳細<br><br>y = ファイルシステム詳細<br>i = 論理システム詳細<br><br>**メモ:**メモ: 以下の要約オプションは export 専用です。extract 関数ではデータの要約はサポートされていません。<br><br>G = グローバル要約<br>デフォルトでは、グローバル要約はオフです。<br><br>A = アプリケーション要約<br><br>D = ディスク デバイス詳細 (7 章の disk コマンドを参照)。 |

| | | | |
|---|---|---|---|
| | | | Z = lvolume 要約 (7 章の lvolume コマンドを参照)。<br>N = netif 要約 (7 章の netif コマンドを参照)。<br>i = 論理システム要約 |
| gapkcdzntuy<br>GADZNTUY<br>(続き) | | | T= トランザクション要約<br>U = CPU 要約 (7 章の cpu コマンドを参照)。<br>Y = filesystem 要約 (7 章の filesystem コマンドを参照)。 |
| -v | | | 冗長出力レポート フォーマットを生成します。 |
| -f | filename | ,new<br>,append<br>,purge | extract データまたは export データをファイルに送信します。filename が指定されていない場合は、データをデフォルトの出力ファイルに送信します(7 章の output コマンドを参照)。 |
| -ut | | | エクスポートされた DSI ログ ファイル データの日付と時刻を UNIX フォーマットで表示します。 |
| -we | 1.....7 | | export から除外する曜日を指定します。1 は日曜日です(weekdays コマンドの項を参照)。 |
| -xp | xopt | | データを外部フォーマット ファイルにエクスポートします(7 章の export コマンドを参照)。 |
| -xt | xopt | | データをシステム内部フォーマットに抽出します(7 章の extract コマンドを参照)。<br>xopt =<br>*dwmy* (Day Week Month Year)<br>*dwmy*-[offset]<br>*dwmy* [absolute] |
| -xw | week | | 月曜日から日曜日までの 1 週間のデータを抽出します(7 章の weekly コマンドを参照)。 |
| -xm | month | | 暦月のデータを抽出します(7 章の monthly コマンドを参照)。 |
| -xy | year | | 暦年のデータを抽出します(7 章の monthly コマンドを参照)。 |
| -? または ? | | | コマンド ライン構文を表示します。 |

コマンド ラインの引数を評価するときや、コマンド ラインにコマンド オプションを入力するときには、次の規則が適用されます。

- エラーと欠落データは、対応するバッチ モード コマンドとまったく同じように処理されます。すなわち、欠落データには可能な場合デフォルトが使用され、エラーがあるとプログラムは直ちに終了しま

す。

- 引数 -v を使用して冗長モードを有効にしていない場合、コマンドおよびコマンドの実行結果のエコーは無効に設定されます。
- 有効な動作を指定していない場合 (-xp、-xw、-xm、-xy、-xt のいずれか)、すべてのパラメータが処理された後、extract ステータスはその stdin ファイルからコマンドの読み込みを開始します。
- 動作を指定している場合 (-xp、-xw、-xm、-xy、-xt のいずれか)、その他のすべてのパラメータが評価された後、プログラムはコマンド オプションを実行します。この場合、パラメータがリストのどの部分に配置されているかは関係ありません。
- コマンド ラインで動作を指定した場合、extract プログラムはその stdin ファイルから読み込みを行いません。代わりに、次の動作に続いてプログラムが終了します。
  extract -f rxdata -r /var/opt/perf/rept1 -xp d-1 -G

これは、次のように解釈します。

| | |
|---|---|
| -f rxdata | 現在のディレクトリの指定された rxdata ファイルに出力します。 |
| -r rept1 | /var/opt/perf/rept1 ファイルには対象となる export フォーマットが含まれます。 |
| -xp d-1 | 現在の日付から1日引いた日 (前日) のデータをエクスポートします。 |
| -G | グローバル要約データをエクスポートします。 |

実際のエクスポートは最後に行われるため、-G パラメータはエクスポートが行われる前に処理されることに注意してください。

ログファイルが指定されていないため、デフォルトの logglob ファイルを使用していることに注意してください。

動作が指定されているため (-xp)、エクスポートが終了すると、extract プログラムはその stdin ファイルから読み込みを行わずに終了します。さらに、冗長モードが -v コマンド オプションで設定されていないため、stdout に対して無関係なすべての出力が削除されます。

# エクスポート関数の概要

extract プログラムの export コマンドは Performance Collection Component の生ログファイル、抽出ログファイル、DSI ログファイルの各データを、エクスポート ファイルに変換します。export コマンドは、ASCII、データファイル、バイナリ、WK1 (表計算シート) のうちいずれかのフォーマットでファイルを書き込みます。エクスポート ファイルは、レポート、カスタム グラフィックス ソフト、データベース、ユーザーが記述した分析プログラムなど、さまざまな方法で使用できます。

## データのエクスポート方法

データをエクスポートするには、次の方法が最も簡単です。

- データをエクスポートするデフォルトのグローバル ログファイル /var/opt/perf/datafiles/logglob を指定します。

- エクスポート データのフォーマットを定義するデフォルトのエクスポート テンプレート ファイル /var/opt/perf/reptfile を指定します。
- エクスポート関数を開始します。

エクスポート データは、現在のディレクトリにあるデフォルトの出力ファイル xfrdGLOBAL.asc に配置されます。出力ファイルでは、ASCII フォーマットが印刷に適しています。

このデフォルトのデータ セット以外のものをエクスポートする場合は、export コマンドと共にその他のコマンドとファイルを使用します。

次のデータ型をエクスポートできます。

| | |
|---|---|
| global | 5分ごとの要約および1時間ごとの要約 |
| application | 5分ごとの要約および1時間ごとの要約 |
| process | 1分間の詳細 |
| disk device | 5分ごとの要約および1時間ごとの要約 |
| lvolume | 5分ごとの要約および1時間ごとの要約 |
| transaction | 5分ごとの要約および1時間ごとの要約 |
| configuration | データ コレクタの開始時の parm ファイル情報を含む1レコードとシステム構成情報 |
| DSI クラス | DSI ログ ファイルのインターバルと要約 |
| netif | 5分ごとの要約および1時間ごとの要約 |
| cpu | 5分ごとの要約および1時間ごとの要約 |
| filesystem | 5分ごとの要約および1時間ごとの要約 |

- 各データ型に必要なデータ項目 (メトリック) を指定できます。
- データ収集期間の開始日と終了日を、シフトや週末除外フィルタと共に指定できます。
- エクスポート テンプレート ファイルにエクスポート データの対象となるフォーマットを指定します。このファイルは、ASCII (テキスト) フォーマットでファイルを保存できる任意のテキスト エディタまたはワード プロセッサを使用して作成できます。
- デフォルトのエクスポート テンプレート ファイル /var/opt/perf/reptfile も使用できます。このファイルは次の出力フォーマットの設定値を指定します。
  - ASCII ファイル フォーマット
  - 欠落値に対する0 (ゼロ)
  - フィールド区切り記号の空白
  - 60分ごとの要約
  - カラムのヘッダーを含める
  - 指定のデータ型に対応する標準のメトリック セットをエクスポートに含める

**注 :** 特定のプラットフォームで作成したログファイルからデータを抽出またはエクスポートする場合は、同一のプラットフォームの reptall ファイルを使用してください。これは、サポートされるメトリック クラスのリストがプラットフォームによって異なるためです。

# エクスポート タスクのサンプル

Performance Collection Component には、エクスポート テンプレート ファイルの 2 つのサンプル repthist と reptall が用意されています。これらのファイルは /var/opt/perf/ ディレクトリにあります。repthist と reptall を使用すると、共通のエクスポート タスクを実行できます。また、これらのファイルは次に説明するタスクのようなカスタム タスクの開始点としても使用できます。

## 印刷可能な CPU レポートの生成

エクスポート テンプレート ファイル repthist には、ある期間のシステムの CPU とディスク使用量の文字グラフを生成するための指定が含まれます。このグラフは印刷可能な文字から構成され、132 カラムを印刷可能なデバイスに印刷できます。たとえば、次の extract プログラムのコマンドを使用して、最近 7 日間のグラフを生成できます。これは、約 2 ページ分になります (1 時間ごとの要約の代わりに、5 分間の詳細を指定すると、34 ページ分のデータが作成されます)。

```
logfile /var/opt/perf/datafiles/logglob
report /var/opt/perf/repthist
global summary
start today-7
export
```

エクスポート データは、エクスポート ファイル xfrsGLOBAL.asc に書き込まれます。このファイルを印刷するには、次のように入力します。

```
lp xfrsGLOBAL.asc
```

## カスタマイズしたエクスポート ファイルの作成

エクスポート テンプレート ファイルを新規作成する場合は、extract プログラムの guide コマンドを使用してエクスポート テンプレート ファイルをコピーしてカスタマイズします。ガイド モードでは、reptall ファイルを /var/opt/perf/ ディレクトリからコピーし、指定されている scope または DSI のログファイルを読み込み、動的にデータ型とメトリック名のリストを作成します。

reptall ファイルには、scope ログファイル データの各データ型に使用可能なメトリックが含まれているため、必要な作業は使用するメトリックをコメントから外すだけです。この方法は、エクスポート テンプレート ファイル全体を入力し直すより簡単です。

# データ ファイルのエクスポート

出力ファイル名を指定するために、export コマンドを発行する前に output コマンドを使用した場合は、すべてのエクスポート データがこのファイルに追加されます。対話型モードで、extract プログラムを実行し、直接ワークステーション (標準出力ファイル) にデータをエクスポートする場合は、export コマンドを発行する前に、extract コマンドで output stdout を指定します。

出力ファイルがデフォルトに設定されると、エクスポートするデータ型によって、エクスポート データが 14 種類のデフォルトの出力ファイルに分割されます。

デフォルトのエクスポート ログ ファイル名は次のとおりです。

| | |
|---|---|
| xfrdGLOBAL.ext | グローバル詳細データ ファイル |
| xfrsGLOBAL.ext | 1 時間ごとのグローバル要約データ ファイル |
| xfrdAPPLICATION.ext | アプリケーション詳細データ ファイル |
| xfrsAPPLICATION.ext | 1 時間ごとのアプリケーション要約データ ファイル |
| xfrdPROCESS.ext | プロセス詳細データ ファイル |
| xfrdDISK.ext | ディスク デバイス詳細データ ファイル |
| xfrsDISK.ext | 1 時間ごとのディスク デバイス要約データ ファイル |
| xfrdVOLUME.ext | 論理ボリューム詳細データ ファイル |
| xfrsVOLUME.ext | 論理ボリューム要約データ ファイル |
| xfrdNETIF.ext | netif 詳細データ ファイル |
| xfrsNETIF.ext | netif 要約データ ファイル |
| xfrdCPU.ext | CPU 詳細データ ファイル |
| xfrsCPU.ext | CPU 要約データ ファイル |
| xfrdFILESYSTEM.ext | filesystem 詳細データ ファイル |
| xfrsFILESYSTEM.ext | filesystem 要約データ ファイル |
| xfrdTRANSACTION.ext | トランザクション詳細データ ファイル |
| xfrsTRANSACTION.ext | トランザクション要約データ ファイル |
| xfrdCONFIGURATION.ext | 構成データ ファイル |

ここで ext は、asc (ASCII)、bin (バイナリ)、dat (データ ファイル)、wk1 (表計算シート) のいずれかです。

**注 :** export コマンドを発行する前に、エクスポート テンプレート ファイルでデータに適合するデータ型と関連する項目を指定しないと、出力ファイルは作成されません。

## エクスポート テンプレート ファイルの構文

エクスポート テンプレート ファイルには、エクスポート データのフォーマット方法およびエクスポート ファイルに含める内容に応じて、次の情報のすべてまたは一部が含まれます。

```
report      "export file title"

format      [ASCII]
            [datafile]
            [binary]
```

```
            [WK1] or
            [spreadsheet]

headings    [on]
            [off]
separator= "char"
summary=value
missing=value
layout=single | multiple
output=filename
data type datatype
items
```

## パラメータ

| | |
|---|---|
| report | エクスポート ファイルのタイトルを指定します。次の項の「エクスポート ファイルのタイトル」を参照してください。 |
| format | エクスポート データのフォーマットを指定します。 |
| | **ASCII** |
| | ASCII (またはテキスト) フォーマットは、プリンタや端末にファイルをコピーする最適なフォーマットです。フィールドは二重引用符 (") で囲みません。 |
| | **datafile** |
| | datafile フォーマットは、ASCII フォーマットに類似していますが、数値以外のフィールドはすべて二重引用符で囲みます。二重引用符により、カラムの位置合わせが厳密でなくなるため、datafile フォーマットのファイルは直接印刷には適していません。datafile フォーマットは、ほとんどの表計算ソフトおよびグラフィックスソフトにインポートするための最も簡単なフォーマットです。 |
| | **binary** |
| | binary フォーマットは、数値が2進整数として表されるため、よりコンパクトです。変換が最小限で済み、メトリックの精度が最高であるため、このフォーマットはユーザーが記述した分析プログラムへの入力に最も適しています。直接印刷には適しません。 |
| | **WK1 (表計算シート)** |
| | WK1 (表計算シート) フォーマットは、Microsoft Excel、その他の表計算ソフトおよびグラフィックスプログラムと互換性があります。 |
| headings | export ファイルにリストされているメトリックのカラムヘッダーを書き込むかどうかを指定します。headings off を指定すると、カラムヘッダーはファイルに書き込まれません。ファイルの最初のレコードはエクスポート データです。headings on を指定すると、ASCII フォーマットと datafile フォーマットでは、最初のデータレコードの前にエクスポート タイトルと、メトリックの各カラムのカラムヘッダーが書き込まれます。binary フォーマット ファイルのカラムヘッダーには、ファイル内のメトリックの説明が含まれます。WK1 フォーマットには、必ずカラムヘッダーが書き込まれます。 |
| separator | ASCII フォーマットまたは datafile フォーマットのデータの各フィールド間に印刷される文字を指定します。デフォルトの区切り記号は空白ですが、多くのプログラムではフィールドの区切り記号としてカンマが使用されています。区切り記号は、印刷される文字を指定しても印刷されない文字を指定してもかまいません。 |
| summary | 各要約インターバルを分単位で指定します。この値は、要約レコードの各レコードに収められる時間を決定します。デフォルトのインターバルは 60 分です。要約値は 5 ～ 1440 分 (1 日) に設定できます。 |
| missing | 欠落データの代わりに使用するデータ値を指定します。欠落データのデフォルト値は 0 です。欠落データと 0 データを区別するために、別の値を指定できます。次の場合は、データ項目が欠落します。 |

| | |
|---|---|
| | • scope コレクタの特定のバージョンで記録されなかった場合<br>• データ項目が属するインスタンス (application、disk、transaction、netif) がそのインターバルにアクティブでなかったために、データ項目が scope で記録されなかった場合<br>• DSI ログファイルの場合、あるインターバルで dsilog プログラムにデータが提供されず、「欠落レコード」となったとき<br>デフォルトでは、欠落レコードはエクスポート データから除外されます。 |
| layout | application、disk、transaction、lvolume、netif などのデータ型に対して、単一のレイアウトか複数のレイアウトか (レコード出力ごとの出力) を single または multiple で指定します。<br>単一のレイアウトは、特定のインターバルでアクティブであった各アプリケーションについて 1 レコードにつき 1 つのインスタンスを書き込みます。例:1 レコードには、エクスポートされたディスクが 1 つ書き込まれます。<br>複数のレイアウトは、各インターバルで 1 レコードに複数のインスタンスを書き込みます。このレコードの一部は、構成されている各アプリケーションについて保存されます。たとえば、1 レコードには、エクスポートされたすべてのディスクが書き込まれます。 |
| output | エクスポートするデータが書き込まれる出力ファイル名を指定します。エクスポートデータ項目のリストを開始するデータ型を示す行の直後に output filename を置くことで、エクスポートする各クラスまたは各 data type について output を指定できます。任意の有効なファイル名を output に指定できます。<br>また、output コマンドを使用すれば名前を対話式に指定できます。名前を指定すると、デフォルトの出力ファイル名がオーバーライドされます。 |
| data type | エクスポート可能なデータ型 (global、application、process、disk、transaction、lvolume、netif、configuration、DSI クラス名) の 1 つを指定します。このパラメータは、このデータ型がエクスポートされるときに、コピーされるデータ項目をリストするエクスポート テンプレート ファイルのセクションを開始します。 |
| items | エクスポート ファイルに書き込むメトリックを指定します。結果ファイルにメトリック名をリストする順序で、1 行に 1 つずつメトリック名をリストします。items をリストする前に、適切な data type を指定する必要があります。エクスポート テンプレート ファイルには、対象となるデータ型の数だけ項目リストを書き込めます。各データ型は、そのデータ型のエクスポートを選択した場合にのみ参照されます。 |

output パラメータおよび layout パラメータは、エクスポート テンプレート ファイル内で複数回使用できます。たとえば、次のようになります。

```
data type global
   output=myglobal
   gbl_cpu_total_util

data type application
   output=myapp
   layout=multiple
   app_cpu_total_util
```

エクスポート テンプレート ファイルは、システム上に複数あってもかまいません。各ファイルが、特定の用途に合うエクスポート ファイル フォーマットのセットを定義できます。report コマンドを使用すると、export 関数で使用するエクスポート テンプレート ファイルを指定できます。

**注:** 1 つのデータ型で異なるレイアウトは指定できません。たとえば、同じ export ファイル内で、data type disk を layout = multiple で指定した後に、layout = single で指定することはできません。

## エクスポート ファイルのタイトル

次の項目は、export file title 文字列で置き換えることができます。

| | |
|---|---|
| !date | export 関数を実行した日付 |
| !time | export 関数を実行した時刻 |
| !logfile | ソース ログ ファイルの完全修飾名 |
| !class | 必要なデータ型 |
| !collector | コレクタ プログラムの名前とバージョン(DSI ログ ファイルでは無効) |
| !system_id | データを収集したシステムの識別子(DSI ログ ファイルでは無効) |

たとえば、次のようになります。

```
report "!system_id data from !logfile on !date !time"
```

この文字列は次のタイトルのようなエクスポート ファイル タイトルを生成します。

```
barkley data from logglob on 02/02/99 08:30 AM
```

# カスタム グラフまたはレポートの作成

エクスポートしたグローバル データとアプリケーション データを含むカスタム グラフまたはレポートを作成するには、次の手順に従います。

1. 各データ型から必要なデータ項目 (メトリック) を選択し、それらにアクセスするときのフォーマットを決定します。
   この例では、ヘッダーがなく、フィールドがカンマで区切られる ASCII ファイルを指定します。
2. 次の ASCII エクスポート テンプレート ファイルを作成し、/var/opt/perf/ ディレクトリに保存します。ファイル名は report1 にします。

```
REPORT "sample export template file (report1)"
FORMAT ASCII
HEADINGS OFF

DATA TYPE GLOBAL
   GBL_CPU_TOTAL_UTIL
   GBL_DISK_PHYS_IO_RATE

DATA TYPE APPLICATION
```

```
APP_CPU_TOTAL_UTIL
APP_DISK_PHYS_IO_RATE
APP_ALIVE_PROCESSES
```

3. extract プログラムを実行します。
4. report コマンドを発行して、作成したエクスポート テンプレート ファイルを指定します。

   **report /var/opt/perf/report1**

5. global コマンドとapplication コマンドを使用して、グローバル要約データとアプリケーション要約データを指定します。

   **global summaryapplication summary**

6. export コマンドを発行して、エクスポートを開始します。

   **export**

7. パフォーマンス データの取得元をプログラムに指定していないため、指定するように促すプロンプトが表示されます。この例では、デフォルトのログ ファイルが正しいため、Enter キーを押すだけです。
8. 次のように出力されます。

```
exporting global data ........50%......100%
exporting application data .....50%......100%

The exported file contains 31 days of data from 01/01/99 to
01/31/99

                          examined  exported
data type                  records   records     space
-----------------------  --------- --------- ---------
global      summaries                    736   0.20 Mb
application summaries                   2560   0.71 Mb
                                             ---------
                                               0.91 Mb
```

作成した2つのファイル(xfrsGLOBAL.asc および xfrsAPPLICATION.asc)に、指定のフォーマットでグローバル要約データとアプリケーション要約データが書き込まれます。

## エクスポート ファイルの出力

各エクスポート ファイルの内容は、次のとおりです。

| | |
|---|---|
| export title line | export title とheadings on を指定した場合 |
| Names (application、netif、lvolume、transaction のいずれか) | 複数レイアウト ファイルで headings on を指定した場合 |
| Heading line1 | headings on を指定した場合 |
| Heading line2 | headings on を指定した場合 |
| first data record | |

| | |
|---|---|
| second data record | |
| ... | |
| last data record | |

レポート タイトルとヘッダーの行はファイル内で繰り返されません。

## ASCII フォーマットと datafile フォーマットに関する注意

これらのフォーマット ファイルのデータは、印刷可能な ASCII フォーマットです。datafile フォーマットで数値以外のすべてのフィールドが二重引用符で囲まれることを除いて、ASCII フォーマットと datafile フォーマットは同一です。datafile では、ヘッダー情報も二重引用符で囲まれます。

ASCII ファイル フォーマットでは、二重引用符でフィールドを囲みません。したがって、ASCII ファイルのデータは、印刷したときにフィールドの位置が揃っています。

数値は、その範囲と内部的な精度に基づいてフォーマットされます。すべてのフィールドの長さが同じとは限らないため、各フィールドを開始するときに使用する区切り記号を指定する必要があります。

ユーザーが指定した区切り記号 (またはデフォルトの空白) は、ASCII フォーマットおよび datafile フォーマットで各フィールドを区切ります。レポートを印刷する場合、区切り記号として空白を使用すると目立ちます。別のプログラムでエクスポート テンプレート ファイルを読み込む場合、別の文字を区切り記号として使用する方が便利です。

コンマは多くのアプリケーションで区切り記号として適用されますが、区切り記号ではないカンマを含むデータ項目もあります。このようなカンマにより、分析プログラムは混乱します。日付と時刻のフォーマットには、extract プログラムを実行するときに指定する母国語に基づき、異なる特殊文字が含まれます。

**注 :** 印刷されない特殊文字を区切り記号として使用するには、エクスポート テンプレート ファイルで separator パラメータの最初の二重引用符の直後にその特殊文字を入力します。

**ヒント :**

- ほとんどの場合、separator="," を使用した datafile フォーマットのファイルは、表計算ソフトで使用できます。
- 多くの表計算ソフトでは、1 シートで最大 256 カラムを使用できます。複数レイアウトのデータ型をエクスポートするときには、合計で 256 を超える項目が生成される可能性が高いので注意してください。report reportfile, show コマンドの出力を使用すると、このような問題が生じるかを特定できます。
- 下線を引けるプリンタの場合、ASCII フォーマットと縦線 (separator=|) を指定し、下線を有効に設定してファイルを印刷すると、見やすくなります。

| | |
|---|---|

# binary フォーマットに関する注意

binary フォーマット ファイルでは、数値が 32 ビット整数として書き込まれます。これにより、ファイル全体のサイズを小さくしてスペースを節約できますが、binary ファイルを読み取れるプログラムが必要になります。binary フォーマット ファイルはプリンタや端末にコピーしないようにしてください。

binary フォーマットでは、数値以外のデータが ASCII フォーマットの場合と同じように書き込まれます。ただし、区切り記号は使用されません。binary フォーマット ファイルを正しく使用するには、report *reportfile*, show を指定するときに、extract で出力されるレコード レイアウト レポートを使用する必要があります。このレポートは、指定した各項目の開始バイトを提供します。

最大精度を維持し、非標準の binary 浮動小数点表現を避けるために、すべての数値が 32 ビット整数として書き込まれます。項目によっては、整数フォーマットに切り捨てる前に定数をかけます。

たとえば、CPU が使用された秒数は、切り捨てる前に 1000 をかけます。エクスポート ファイルの値を実際の秒数に変換するには、その値を 1000 で割ります。簡単に変換できるように、headings on を指定して、エクスポート ファイルに倍率を書き込みます。レポート タイトルと特別なヘッダー レコードは binary フォーマット ファイルに書き込まれ、プログラムによる解釈に役立てられます。

2 進整数は、extract プログラムを実行するシステムに固有のフォーマットで書き込まれます。たとえば、Intel のシステムは、「little endian」整数を書き込みますが、HP-UX、IBM AIX、Sun の各システムは、「big endian」整数を書き込みます。異なる「endian」を使用するシステムにバイナリ エクスポート ファイルを送信するときは、注意してください。

## バイナリ ヘッダー レコード レイアウト

binary フォーマット エクスポート ファイルの各レコードには、ユーザーが指定したデータの前に特殊な 16 バイトのレコード ヘッダーが含まれます。report *reportfile*, show コマンドには、このレコード ヘッダーを構成する次の 4 つのフィールドが含まれます。

## Binary Title Record

BINARY ファイルのタイトル レコードには、エクスポート ファイルの内容をプログラムが解釈するための情報が含まれます。このレコードは、headings on を指定している場合、エクスポート ファイルに書き込まれます。

Binary Title Record の内容は、次のとおりです。

```
Record Length          4 byte Int    Length of Title Record

Record ID              4 byte Int     -1

Date_Seconds           4 byte Int     Date exported file was created

Number_of_vars         4 byte Int     Maximum number of repeating
                                      variables

Size of Fixed Area     4 byte Int     Bytes in nonvariable part of
                                      record

Size of Variable Entry 4 byte Int     Bytes in each variable entry

GMT Time Offset        4 byte Int     Seconds offset from Greenwich
```

```
                                        Mean Time
Daylight Savings Time  4 byte Int       =1 indicates Daylight Savings
                                        Time
System ID             40 Characters,    System Identification
Collector Version     16 Characters,    Name & version of the data
                                        collector
Log File Name         72 Characters,    Name of the source log file
Report Title          100 Characters,   User specified report title
```

Binary Title Record の `Date_Seconds`、`GMT Time Offset`、`Daylight Savings Time` の各メトリックは、システムおよび `export` ファイルが作成された時刻に適用されます。このシステムがデータを記録したシステムと同じでない場合、これらのフィールドはファイル内のデータを正しく反映できません。

## Binary Item Identification Record

バイナリファイルの最初のヘッダーレコード (Record ID -2) には、エクスポートされた各項目に固有の項目番号が含まれます。各項目 ID は 4 バイトの整数で、この製品に用意されている `rxitemid` ファイルを使用して相互に参照できます。項目 ID のフィールドは、ファイルの残りの部分でそれらが示されるデータフィールドに揃えられます。すべてのバイナリエクスポート データ項目が、エクスポート ファイルで 4 バイトを複数回使用し、それぞれが 4 バイトの境界から始まります。データ項目に 4 バイト以上のスペースが必要な場合、対応する項目 ID フィールドの左側が 0 で埋められます。

たとえば、プロセス メトリック Program には 16 バイトが必要です。そのデータレコードと項目 ID レコードは次のようになります。

```
Header 1 (Item Id Record) ...|   0|   0|   0|12012|
Process Data Record          |Prog|ram_|name| _aaa|
```

## Binary Scale Factor Record

バイナリファイルの 2 番目のヘッダーレコード (Record ID -3) には、各エクスポート項目の倍率が含まれます。別のコンピュータアーキテクチャが浮動小数点を実装している場合に問題を最小にするために、数値項目は 32 ビット (4 バイト) 整数としてバイナリファイルにエクスポートされます。ほとんどの項目は、固定倍率をかけてから、整数フォーマットに適合させるために切り捨てられます。これにより、倍数を分母として使用して、浮動小数点数は小数部として表現できます。

各倍率は、ほとんどのデータ項目と一致する 32 ビット (4 バイト) 整数です。数値以外のメトリックおよびその他の特殊な値のメトリックを示すために、倍率に対して特殊な値が使用されます。

## 特殊な倍率

ASCII フィールドなど、数値以外のメトリックには、0 という倍率があります。-1 という倍率は発生しないはずですが、発生した場合、extract プログラムの内部エラーが示され、レポートされます。

DATE フォーマットは、フィールドの中で最も影響の小さい 16 ビット (最も右寄りの 16 ビット) の MPE CALENDAR フォーマットです。日付の倍率は 512 です。これを 32 ビット整数にする (512 で割る) と、年が日付の整数部分として、その年の日 (512 で割る) が小数部として分離されます。

TIME は 4 バイトのバイナリフィールドです (時、分、秒、十分の一秒)。時間の倍率は 65536 です。時間を 65536 で割ると、整数部分が (時 * 256) + 分の数字になります。

Date_Seconds 値は、バイナリファイルで処理する方が簡単です。

## Application Name Record

アプリケーション データを Multiple Layout フォーマットでエクスポートすると、特殊な Application Name Record が書き込まれ、アプリケーション グループを識別します。バイナリフォーマット ファイルの場合、このレコードには Record ID -4 が含まれます。このレコードは、バイナリレコードのヘッダー (16 バイト) と、エクスポート データの開始日に定義された各アプリケーション名 (20 バイト) から構成されます。

データ抽出時間にアプリケーションが追加または削除される場合、Application Name Record は新しいアプリケーション名で繰り返されます。

## Transaction Name Record

トランザクション データを Multiple Layout フォーマットでエクスポートすると、特殊な Transaction Name Record が書き込まれ、トランザクション名を識別します。バイナリフォーマット ファイルの場合、このレコードには Record ID -5 が含まれます。このレコードは、バイナリレコードのヘッダー (16 バイト) と、エクスポート データの開始日に構成された各トランザクションのアプリケーショントランザクション名 (60 バイトに切り捨て) から構成されます。データ抽出時間にトランザクションが追加される場合、Transaction Name Record は、元のリストの最後に新しいアプリケーショントランザクション名が追加されて繰り返されます。データ抽出開始後に削除されるトランザクションは、Multiple Layout データレコードから削除されません

## Disk Device Name Record

ディスク デバイス データを Multiple Layout フォーマットでエクスポートすると、特殊な Disk Device Name Record が書き込まれ、ディスク デバイス名を識別します。バイナリフォーマット ファイルの場合、このレコードには Record ID -7 が含まれます。このレコードは、バイナリレコードのヘッダー (16 バイト) と、エクスポート データの開始日に構成された各ディスク デバイス名 (20 バイト) から構成されます。

データ抽出時間にディスク デバイスが追加される場合、Disk Device Name Record は、元のリストの最後に新しいディスク デバイス名が追加されて繰り返されます。データ抽出開始後に削除されるディスク デバイスは、Multiple Layout データレコードから削除されません。

## Logical Volume Name Record

論理ボリューム データを Multiple Layout フォーマットでエクスポートすると、特殊な Logical Volume Name Record が書き込まれ、論理ボリューム名を識別します。バイナリフォーマット ファイルの場合、このレコードには Record ID -8 が含まれます。このレコードは、バイナリレコードのヘッダー (16 バイト) と、エクスポート データの開始日に構成された各論理ボリュームのディスク デバイス名 (20 バイト) から構成されます。

データ抽出時間に論理ボリュームが追加される場合、Logical Volume Name Record は、元のリストの最後に新しい論理ボリューム名が追加されて繰り返されます。データ抽出開始後に削除される論理ボリュームは、Multiple Layout データレコードから削除されません。

## Netif Name Record

`netif` データを Multiple Layout フォーマットでエクスポートすると、特殊な Netif Name Record が書き込まれ、netif デバイス名を識別します。バイナリフォーマット ファイルの場合、このレコードには Record ID -11 が含まれます。このレコードは、バイナリレコードのヘッダー (16 バイト) と、エクスポート データの開始日に構成された各 `netif` デバイス名の 20 バイトの `netif` デバイス名 から構成されます。

データ抽出時間に `netif` デバイスが追加される場合、Netif Name Record は、元のリストの最後に新しいデバイス名が追加されて繰り返されます。データ抽出開始後に削除される netif デバイスは、Multiple Layout データレコードから削除されません。

# extract のコマンド

この章 では、**extract** プログラムのコマンド について説 明します。この章 は、コマンド 構 文 をまとめた表 、データの抽 出 およびエクスポートを行 うコマンド の表 、アルファベット 順 にコマンド を説 明しているコマンド リファレンスから構 成 されています。

**extract** のコマンド とパラメータは、大 文 字 と小 文 字 を組 み合 わせて入 力 できます。コマンド 名 の最 初 の 3 文 字 だけは必 ず入 力 します。ただし、weekdays コマンド とweekly コマンド だけは、コマンド 名 全 体 を入 力 する必 要 があります。たとえば、application detail コマンド は app det と省 略 できます。

これらのコマンド の使 用 例 については、**extract** プログラムのオンライン ヘルプを参 照 してください。

次 のページの表 では、**extract** コマンド とパラメータの構 文 をまとめています。

**メモ :**extract 関 数 では要 約 データは作 成 できません。要 約 データは export 関 数 でのみ作 成 することができます。

**表 7: extract のコマンド: 構文とパラメータ**

| コマンド | パラメータ |
|---|---|
| `application` | ondetailsummary<br><br>(**export** のみ)<br>both (**export** のみ)<br>off (デフォルト ) |
| `class` | detail (デフォルト )<br>summary (**export** のみ)<br>both (**export** のみ)<br>off |
| `cpu` | detailsummary<br>(**export** のみ)<br>both (**export** のみ)<br>off (デフォルト ) |
| `configuration` | `ondetailoff`<br><br>(デフォルト )。 |
| `disk` | `ondetailsummary`<br><br>(**export** のみ)<br>`both` (**export** のみ)<br>`off` (デフォルト ) |
| `exite` | |

| コマンド | パラメータ |
|---|---|
| `export` | day[*ddd*] [*-days*]<br>week [*ww*] [*-weeks*]<br>month[*mm*] [*-months*]<br>year [*yy*] [*-years*] |
| `extract` | day[*ddd*] [*-days*]<br>week [*ww*] [*-weeks*]<br>month[*mm*] [*-months*]<br>year [*yy*] [*-years*] |
| `filesystem` | detailsummary<br>(**export** のみ)<br>both (**export** のみ)<br>off (デフォルト) |
| `global` | ondetail<br>(デフォルト)。<br>summary (**export** のみ)<br>both (**export** のみ)<br>off |
| `guide` | |
| `help` | |
| `list` | `filename *` |
| `logfile` | `logfile` |
| `lvolume` | ondetail summary<br><br>(**export** のみ)<br>both (**export** のみ)<br>off (デフォルト) |
| `menu` | |
| `monthl` | |
| `y` | *yyymmmm* |
| `netif` | ondetail summary<br><br>(**export** のみ)<br>both (**export** のみ)<br>off (デフォルト) |
| `output` | *outputfile*<br>,new ,purgeboth ,append |

| コマンド | パラメータ |
|---|---|
| `process` | `ondetail [app#[-#],...]off` (デフォルト)`killed` |
| `quitq` | |
| `report` | [*export template file*], show |
| `shift` | *starttime* - *stoptime*<br>all day noweekends |
| `sh!` | シェル コマンド |
| `show` | `all` |
| `start` | *date* [*time*]<br>today [-*days*] [*time*]<br>last [-*days*] [*time*]<br>first [+*days*] [*time*] |
| `stop` | *date* [*time*]<br>today [-*days*] [*time*]<br>last [-*days*] [*time*]<br>first [+*days*] [*time*] |
| `transaction` | ondetail summary<br><br>(**export** のみ)<br>both (**export** のみ)<br>off (デフォルト) |
| `weekdays` | `1.....7` |
| `weekly` | *yywwww* |
| `yearly` | *yyyyyy* |

次 の表 は、データの抽 出 およびエクスポートに使 用 するコマンド と、使 用 されるログ ファイル (scope ログ ファイルまたは DSI ログ ファイル) の種 類 のリスト です。

**表 8 extract のコマンド : データの抽 出 とエクスポート**

| コマンド | データの抽出 | データのエクスポート | scope ログ ファイル | DSI ログ ファイル |
|---|---|---|---|---|
| `application` | x | x | x | |
| `class` | x | x | x | x |
| `configuration` | | x | x | |

| コマンド | データの抽出 | データのエクスポート | scope ログファイル | DSI ログファイル |
|---|---|---|---|---|
| cpu | x | x | x | |
| disk | x | x | x | |
| export | | x | x | x |
| extract | x | | x | |
| filesystem | x | x | x | |
| global | x | x | x | |
| logfile | x | x | x | x |
| lvolume | x | x | x | |
| monthly | x | | x | |
| netif | | x | x | |
| output | x | x | x | x |
| process | x | x | x | |
| report | | x | x | x |
| shift | x | | x | x |
| start | x | x | x | x |
| stop | x | x | x | x |
| transaction | x | x | x | x |
| weekdays | | x | x | x |
| weekly | x | | x | |
| yearly | x | | x | |
| logicalsystems | x | x | x | |

# application

application コマンドは、抽出またはエクスポートするアプリケーション データの種類を指定するときに使用します。

デフォルトは、application off です。

**構文**

| | |
|---|---|
| **application** | **[on] [detail] [summary] [both] [off]** |

| | |
|---|---|
| | |

**パラメータ**

| | |
|---|---|
| on または detail | 生の 5 分間詳細データの抽出またはエクスポートを指定します。 |
| summary<br>(**export** のみ) | 次の基準によるデータの要約を指定します。<br>指定したエクスポート テンプレート ファイルの summary パラメータで指定される時間 (分単位) (**export** のみ)<br>デフォルトの要約インターバル (1 時間) (**export** または **extract**)<br>使用する要約インターバルによりますが、抽出またはエクスポートしたデータのサイズは要約によって大幅に減少できます。たとえば、1 時間ごとの要約データは、5 分間の詳細データのサイズの約 10 分の 1 になります。 |
| both<br>(**export** のみ) | 詳細データおよび要約データの抽出またはエクスポートを指定します。 |
| `off` | このタイプのデータを抽出またはエクスポートしないことを指定します。 |

**メモ:** Performance Manager を使用している場合、5 分間隔のポイントでアプリケーション グラフを作成する前に、抽出ファイルに詳細データを含める必要があります。

**例**

この例では、application コマンドにより、詳細アプリケーション ログ ファイル データがエクスポートされます。**output export** ファイルには、**myrept** エクスポート テンプレート ファイルで指定されているアプリケーション メトリックが含まれます。

```
logfile /var/opt/perf/datafiles/logglob

global off

application detail

report /var/opt/perf/myrept

export
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -a -r /var/opt/perf/myrept -xp
```

# class

class コマンドは、エクスポートする DSI データのクラスや、抽出またはエクスポートする scope データのクラスを指定するときに使用します。

デフォルトは、class detail です。

**構文**

| | | |
|---|---|---|
| | | **[detail]** |
| **class** | [*classname*] | **[summary]** |
| | | **[both]** |
| | | **[off]** |

**パラメータ**

| | |
|---|---|
| classname | 同類をまとめたメトリックのグループ名です。 |
| detail | DSI ログファイルの場合は、DSI ログファイルに設定されている時刻に従ってエクスポートする詳細データの量を指定します詳細については、「データソース統合の概要」を参照してください。<br>scope ログファイルの場合は、生の 5 分間の詳細を抽出またはエクスポートすることを指定します。 |
| summarybothoff | application コマンドの説明にある「パラメータ」を参照してください。summary および both はエクスポートのみ可能です。 |

**例**

クラス acctg_info を含む DSI ログファイルに要約データをエクスポートするには、次のコマンドを発行します。

```
class acctg_info summary
```

ユーザーがログファイルを指定し、 **extract** プログラムによりこのファイルをオープンすると、acctg_info クラスがログファイルにあることが確認され、このクラスがエクスポートされます。

このコマンドの別の使用例を示します。

```
CLASS ACCTG_INFO SUMMARYclass ACCTG_INFO summaryclass acctg_info sum
```

コマンドは大文字でも小文字でもかまいません。クラス名は常に大文字に変換され、比較されます。

次の例では、クラス fin_info の要約データをエクスポートします。

```
extract>
class fin_info summary
export
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -l dsi.log -C fin_info summary -xp
```

# configuration

configuration コマンドは、システム構成情報をエクスポートするかどうかを指定するときに使用します。

デフォルトは、configuration off です。

**構文**

| | |
|---|---|
| **configuration** | **[on] [detail] [off]** |

**パラメータ**

| | |
|---|---|
| on または detail | すべての構成レコードのエクスポートを指定します。 |
| off | 構成データをエクスポートしないことを指定します。 |

ログファイルに含まれるすべての構成情報がエクスポートされます。 configuration コマンドと共に使用される begin、 end、 shift、 start、 stop、noweekends の各コマンドはすべて無視されます。

**メモ :**configuration コマンドは、export 関数のみに影響します。extract 関数は必ずシステム構成情報を抽出するため、extract 関数には影響しません。

**例**

この例では、configuration コマンドにより、システム構成情報がエクスポートされます。出力エクスポート ファイルには、 **myrept** エクスポート テンプレート ファイルで指定されている構成メトリックが含まれます。

```
logfile /var/opt/perf/datafiles/logglob
configuration on
report /var/opt/perf/myrept
export
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -c -r /var/opt/perf/myrept -xp
```

# cpu

cpu コマンドは、CPU の要約レベルを指定するときに使用します。

デフォルトは、cpu off です。

**構文**

| | |
|---|---|
| **cpu** | **[detail] [summary] [both]**<br>**[off]** |

**パラメータ**

| | |
|---|---|
| detail | 5 分間の詳細レコードを抽出またはエクスポートします。 |
| summary | 要約レコードをエクスポートします。 |
| both | 詳細レコードと要約レコードの両方をエクスポートします。 |
| off | CPU データを抽出またはエクスポートしません。 |

**例**

この例では、cpu コマンドにより、2001 年 7 月 26 日から収集された CPU detail データがエクスポートされます。エクスポート テンプレート ファイルを指定していないため、デフォルトのエクスポート テンプレート ファイル **reptfile** が使用されます。**reptfile** で指定されているように、すべてのディスク メトリックが出力ファイルに含まれます。

```
logfile /var/opt/perf/datafiles/logglob

global off

cpu detail

start 7/26/01

export
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -u -b 7/26/01 -xp
```

# disk

disk コマンドは、抽出またはエクスポートするディスク デバイス データの種類を指定するときに使用します。

デフォルトは、disk off です。

**構文**

| | |
|---|---|
| **disk** | **[on][detail][summary][both][off**<br><br>] |

**パラメータ**

| | |
|---|---|
| on または detailsummaryboth<br><br>off | この章の最初に説明されている「application」(137ページ) コマンドの説明にある「パラメータ」を参照してください。summary および both はエクスポートのみ可能です。 |

**例**

この例では、disk コマンドにより、1999 年 7 月 5 日から収集された disk detail データがエクスポートされます。エクスポート テンプレート ファイルを指定していないため、デフォルトのエクスポート テンプレート ファイル **reptfile** が使用されます。**reptfile** で指定されているように、すべてのディスク メトリックが出力ファイルに含まれます。

```
logfile /var/opt/perf/datafiles/logglob

global off

disk detail

start 7/5/99
```

`export`

コマンド ライン引 数を使 用して上 記 の操 作を実 行 するには、次 のように入 力します。

**`extract -D -b 7/5/99 -xp`**

# exit

exit コマンド は、**extract** プログラムを終 了 するときに使 用します。quit コマンド は、**extract** プログラムの exit コマンド に相 当します。

**構 文**

**`exit`**

**`e`**

# export

export コマンド は、エクスポート ファイル フォーマット にデータをコピーするプロセスを開 始 するときに使 用します。

**構 文**

| | | |
|---|---|---|
| **`export`** | **`[day[week[month[year`** | **[***ddd***] [***yyddd***] [-***days***]][***<br>***ww***] [***yyww***] [-***weeks***]][***<br>***mm***] [***yymm***][-***months***]][***<br>***yy***] [***yyyy***] [-***years***]]** |

**パラメータ**

特 定 のインターバルでデータをエクスポートするときには、次 のパラメータの 1 つを使 用します。

| | |
|---|---|
| `day` | 1 日を表します |
| `week` | 月 曜 日 から<br>日 曜 日 までの 1 週 間を表します |
| `month` | 月 の最 初 の日 から最 後 の日 までの<br>暦 年を表します |
| `year` | 年 の最 初 の日 から最 後 の日 までの<br>暦 年を表します |

export コマンド と共 にパラメータを使 用しない場 合 は、エクスポート データに対して使 用 するインターバルが、start コマンド および stop コマンド により設 定 されます。

**使 い方**

特 定 のインターバル (day、week、month、year) を指 定 するには、次 の 4 つの方 法 があります。

- 現在のインターバル – パラメータのみを指定します。たとえば、month は当月を意味します。
- 過去のインターバル – パラメータ、マイナス符号、現在からさかのぼるインターバルの数を指定します。たとえば、day-1 は前日を示し、week-2 は今週から2週間前の週を示します。
- 絶対インターバル – パラメータと正数を指定します。数字は、現在の年の絶対インターバルを示します。たとえば、day 2 は今年の1月 2日を示します。
- 絶対インターバルと年 – パラメータと大きな整数を指定します。数字は、年の最後の2桁とその年の絶対インターバルを示す数から構成する必要があります。このフォーマットでは、絶対日付には5桁を使用し (99002 は 1999 年 1月 2日を示します)、その他のすべてのパラメータでは4桁を使用します (month 9904 は 1999 年 4月を示します)。

事前にログファイルまたはエクスポート テンプレート ファイルを指定していない場合、logfile コマンドはデフォルトのグローバル ログ ファイル **logglob** ファイルを使用し、report コマンドはデフォルトのエクスポート テンプレート ファイル **reptfile** を使用します。

その他のすべてのパラメータでは、設定値またはデフォルトが使用されます。これらの動作の詳細は、「application」、「configuration」、「global」、「process」、「disk」、「lvolume」、「netif」、「CPU」、「filesystem」、「transaction」、「output」、「shift」、「start」、「stop」のコマンド説明を参照してください。

export コマンドは、指定したデータ型および要約レベルに基づき、最大 16 種類のデフォルトの出力ファイルを作成します。

| | |
|---|---|
| **xfrdGLOBAL.ext** | グローバル詳細データ ファイル |
| **xfrsGLOBAL.ext** | 1時間ごとのグローバル要約データ ファイル |
| **xfrdAPPLICATION.ext** | アプリケーション詳細データ ファイル |
| **xfrsAPPLICATION.ext** | 1時間ごとのアプリケーション要約データ ファイル |
| **xfrdPROCESS.ext** | プロセス詳細データ ファイル |
| **xfrdDISK.ext** | ディスク デバイス詳細データ ファイル |
| **xfrsDISK.ext** | ディスク デバイス要約データ ファイル |
| **xfrdVOLUME.ext** | 論理ボリューム詳細データ ファイル |
| **xfrsVOLUME.ext** | 論理ボリューム要約データ ファイル |
| **xfrdNETIF.ext** | netif 詳細データ ファイル |
| **xfrsNETIF.ext** | netif 要約データ ファイル |
| **xfrdCPU.ext** | CPU 詳細データ型 |
| **xfrsCPU.ext** | CPU 要約データ型 |
| **xfrdFILESYSTEM.ext** | filesystem 詳細データ型 |
| **xfrsFILESYSTEM.ext** | filesystem 要約データ型 |

| | |
|---|---|
| **xfrdTRANSACTION.ext** | トランザクション詳細データファイル |
| **xfrsTRANSACTION.ext** | トランザクション要約データファイル |
| **xfrdCONFIGURATION.ext** | 構成詳細データファイル |

ここで、 **ext** は **asc**、**dat**、**bin**、**wk1** のいずれかです

デフォルトのファイル名は、データ型名から作成されます。データが詳細データであるか要約データであるかによって、接頭辞が **xfrd** または **xfrs** になります。拡張子は、指定されているデータフォーマットによって、**asc** (ASCII)、**bin** (バイナリ)、**dat** (データファイル)、 **wk1** (表計算シート) のいずれかになります。

たとえば、classname = ACCTG_INFO の場合は、次の **export** ファイル名になります。

| | |
|---|---|
| **xfrdACCTG_INFO.wk1** | ACCT_INFO の詳細表計算シート データ |
| **xfrsACCTG_INFO.asc** | ACCT_INFO の要約 ASCII データ |

データのエクスポートに関する詳細は、6 章の「エクスポート関数の概要」を参照してください。

**例**

この例では、export コマンドにより、前日の 8:00 AM から 5:00 PM までに収集されたログファイルデータをエクスポートします。エクスポート テンプレート ファイルを指定していないため、デフォルトのエクスポート テンプレート ファイル **reptfile** が使用されます。**reptfile** で指定されているように、すべてのグローバル メトリックが出力ファイルに含まれます。

```
logfile /var/opt/perf/datafiles/logglob

start today-1 8:00 am

stop today-1 5:00 pm

global both

export
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -gG -b today-1 8:00 am -e today-1 5:00 pm -xp
```

# extract

extract コマンドは、生ログファイルから抽出ファイルフォーマットにデータをコピーするプロセスを開始するときに使用します。抽出ファイルは、アーカイブするときや、Performance Manager などの分析プログラムで分析するときに使用できます。データを生ログファイルと抽出ファイルから抽出できます。

extract コマンドは、DSI ログファイルのデータを処理するときには使用できません。

**構文**

| | | |
|---|---|---|
| **extract** | **[day[week[month[year** | **[***ddd***] [***yyddd***] [-***days***]][***ww***] [***yyww***] [-***weeks***]][** |

| | | |
|---|---|---|
| | | *mm*] [*yymm*][-*months*]][ *yy*] [*yyyy*] [-*years*]] |

**パラメータ**

特定のインターバルでデータを抽出するときには、次のパラメータの 1 つを使用します。

| | |
|---|---|
| day | 1 日を表します |
| week | 月曜日から日曜日までの 1 週間を表します |
| month | 月の最初の日から最後の日までの暦月を表します |
| year | 年の最初の日から最後の日までの暦年を表します |

extract コマンドと共にパラメータを使用しない場合は、データ抽出に対して使用するインターバルが、start コマンドおよび stop コマンドにより設定されます。

**使い方**

特定のインターバル (day、week、month、year) を指定するには、次の 4 つの方法があります。

- 現在のインターバル ー パラメータのみを指定します。たとえば、month は当月を意味します。
- 過去のインターバル ー パラメータ、マイナス符号、現在からさかのぼるインターバルの数を指定します。たとえば、day-1 は前日を示し、week-2 は今週から 2 週間前の週を示します。
- 絶対インターバル ー パラメータと正数を指定します。数字は、現在の年の絶対インターバルを示します。たとえば、day 2 は今年の 1 月 2 日を示します。
- 絶対インターバルと年 ー パラメータと大きな整数を指定します。数字は、年の最後の 2 桁とその年の絶対インターバルを示す数から構成する必要があります。このフォーマットでは、日付には 5 桁を使用し (99002 は 1999 年 1 月 2 日を示します)、その他のすべてのパラメータでは 4 桁を使用します (month 99904 は 1999 年 4 月を示します)。

extract コマンドがデータの抽出を開始します。事前に指定していない場合、logfile コマンドと output コマンドは、`extract` コマンドが実行されるときに次のデフォルトを仮定します。

```
log file = /var/opt/perf/datafiles/logglob
```

```
output file = rxlog,new
```

その他のすべてのパラメータでは、設定値またはデフォルトが使用されます。これらの動作の詳細は、「application」、「global」、「process」、「disk」、「lvolume」、「netif」、「CPU」、「filesystem」、「transaction」、「shift」、「start」、「stop」のコマンド説明を参照してください。

抽出するログファイルのサイズは、最大 3.5 ギガバイトです。

**例**

最初の例では、2000 年 3 月 1 日から 2000 年 6 月 30 日の平日の 8:00 AM から 5:00 PM までに収集されたデータが抽出されます。グローバル要約データとアプリケーション詳細データだけが抽出されます。

```
logfile /var/opt/perf/datafiles/logglob
```

```
start 03/01/00
```

```
stop 06/30/00
```

```
shift 8:00 am - 5:00 pm noweekends
global detail
application detail
extract
```

コマンド ライン引 数を使 用して上 記 の操 作を実 行 するには、次 のように入 力します。

```
extract -ga -b 03/01/00 -e 6/30/00 -s 8:00 am - 5:00 noweekends -xt
```

2 番 目 の例 では、新しい抽 出 ログ ファイル **rxjan00** が作 成 されます。この名 前 が付いている既 存 のファイルは削 除 されます。2000 年 1 月 1 日 から 2000 年 1 月 31 日 までに収 集 されたすべての生 ログ ファイル データが抽 出 されます。

```
logfile /var/opt/perf/datafiles/logglob
output rxjan00,purge
start 01/01/00
stop 01/31/00
global detail
application detail
transaction detail
process detail
disk detail
lvolume detail
netif detail
filesystem detail
cpu detail
extract
```

コマンド ライン引 数を使 用して上 記 の操 作を実 行 するには、次 のように入 力します。

```
extract -f rxjan00,purge -gatpdznyu -b 01/01/00 -e 01/31/00 -xt
```

# filesystem

filesystem コマンド は、**extract** または **export** を実 行 する filesystem データの要 約レベルを指 定 するときに使 用します。

デフォルト は、filesystem off です。

**構 文**

| | |
|---|---|
| **filesystem** | **[detail][summary][both][off]** |

**パラメータ**

| | |
|---|---|
| detail | 5分間の詳細レコードを抽出またはエクスポートします。 |
| summary | 要約レコードをエクスポートします。 |
| both | 詳細レコードと要約レコードの両方をエクスポートします。 |
| off | filesystem データを抽出またはエクスポートしません。 |

**例**

この例では、filesystem コマンドにより、2001 年 7 月 26 日から収集された filesystem 詳細データがエクスポートされます。エクスポート テンプレート ファイルを指定していないため、デフォルトのエクスポート テンプレート ファイル **reptfile** が使用されます。**reptfile** で指定されているように、すべての filesystem メトリックが出力ファイルに含まれます。

```
logfile /var/opt/perf/datafiles/logglob
global off
filesystem detail
start 7/26/01
export
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -y -b 7/26/01 -xp
```

# global

global コマンドは、抽出またはエクスポートするグローバル データの量を指定するときに使用します。

デフォルトは、global detail です(コマンド ライン モードでのデフォルトは global off です)。

**構文**

| | |
|---|---|
| **global** | **[on] [detail] [summary] [both] [off]** |

**パラメータ**

| | |
|---|---|
| detail or onsummarybothoff | この章の最初に説明されている「application」(137ページ) コマンドの説明にある「パラメータ」を参照してください。summary および both はエクスポートのみ可能です。 |

**使い方**

5分間隔のポイントで Performance Manager グローバル グラフを作成する場合、詳細データを抽出する必要があります。

グラフの作成に必要なデータレコードが少ないため、要約データのグラフは Performance Manager によって迅速に作成されます。グローバル要約データのみが抽出される場合、Performance Manager のグローバルグラフは 5 分間隔のデータポイントでは作成できません。

both オプションは 1 時間ごとの要約レコードにより得られるアクセス速度を維持します。また、5 分間隔のポイントで Performance Manager グローバルグラフを作成できます。

Performance Manager を使用している場合、システム全体の動作を正しく理解するにはグローバルデータが必要なため、off パラメータを指定しないようにしてください。抽出ファイルに少なくとも 1 種類のグローバルデータが含まれていない場合、Performance Manager グローバルグラフは作成できません。

**例**

この例では、グローバルデータをエクスポートしないことを指定するために、global コマンドが使用されています (global detail がデフォルトです)。詳細トランザクションデータのみがエクスポートされます。**output export** ファイルには、**myrept** エクスポートテンプレートファイルで指定されているトランザクションメトリックが含まれます。

```
extract>

logfile /var/opt/perf/datafiles/logglob

global off

transaction detail

report /var/opt/perf/myrept

export
```

コマンドライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -l -t -r /var/opt/perf/myrept -xp
```

# guide

guide コマンドは、ガイドコマンドモードに移行するときに使用します。ガイドコマンドインターフェイスは extract のさまざまなコマンドについてガイドし、一般的で実行可能ないくつかの作業を行うように促すプロンプトを表示します。

**構文**

```
guide
```

**使い方**

**extract** の対話型モードからガイドコマンドモードに移行するには、**guide** と入力します。

パラメータのデフォルト値を適用するには、**Return** キーを押します。

ガイドコマンドモードを終了し、対話型モードに戻るには、guide> プロンプトで **q** と入力します。

このコマンドでは、パラメータ設定値を組み合わせる必要がありません。このコマンドは、ほとんどのユーザーに有益な結果をもたらす設定値を選択します。**extract** の対話型コマンドモードからは、**extract** の関数を完全に制御できます。

**メモ:** DSI ログ ファイル データをエクスポートしている場 合 、ガイド コマンド モードを使 用して、カスタマイズしたエクスポート テンプレート ファイルを作 成し、データをエクスポートすることをお勧 めします。

# help

help コマンドは、オンライン ヘルプにアクセスするときに使 用します。

**構 文**

**help [***keyword***]**

**使い方**

**extract** のコマンドと作 業 に関 する情 報 (ヘルプの情 報 ) を得 るには、パラメータを入 力します。キーワードを入 力すると、別 のトピックに移 動できます。情 報 が複 数 のページにわたる場 合 は、**Return** を押 すと次 のページが表 示されます。ヘルプ システムを終 了し、**extract** プログラムに戻 るには、**q** または **quit** と入 力します。

また、特 定 のトピックに関 するヘルプも要 求 できます。たとえば、次 のようになります。

```
help tasks
```

または

```
help resize parms
```

この形 式 の help コマンドを使 用 すると、指 定 のトピックのヘルプ テキストを受 信しますが、extract のコマンド エントリコンテキストは終 了していません。対 話 型 モードでヘルプ サブシステムを指 定していないため、次 の extract コマンドを入 力する前 に **quit** を入 力する必 要 はありません。

# list

list コマンドは、extract プログラムのすべてのレポートのリスト ファイルを指 定 するときに使 用します。

**構 文**

| | |
|---|---|
| `list` | **[file]** |
| | **[*]** |

**使い方**

extract を使 用 中 にリスト デバイスを指 定 する場 合 は、随 時 **list** を使 用 できます。このコマンドは、ファイル名 またはリスト デバイス名 を使 用 して、ユーザー指 定 の設 定 値 を出 力します。リスト ファイルがすでに存 在 する場 合 、出 力 はそのファイルに追 加 されます。

リスト デバイスに送 信されるデータも画 面 上 に表 示されます。

**extract** の実 行 中 に、次 のように入 力します。

```
list outfilename
```

リストされているデバイスをユーザー端 末 に返 すには、次 のいずれかを入 力します。

**list stdout**

または

**list ***

現在のリスト デバイスを特定するには、次のようにパラメータを使用せずに list コマンドを入力します。

**list**

リスト ファイルが **stdout** でない場合、ほとんどのコマンドは入力されたときにリスト ファイルにエコーされます。

**例**

次の例では、リスト デバイスが mylist に設定されています。次のコマンドの結果は、mylist に出力され、画面に表示されます。

```
extract>
logfile /var/opt/perf/datafiles/logglob
list mylist
global detail
shift 8:00 AM - 5:00 PM
extract
```

# logfile

logfile コマンドは、ログ ファイルをオープンするときに使用します。ログ ファイルは、`extract` プログラムのすべての関数に対してオープンする必要があります。これを明示的に行うには logfile コマンドを発行し、暗黙的に行うには extract コマンドまたは export コマンドを発行します。ログ ファイル名を指定しない場合は、**extract** プログラムがログ ファイル名の入力を促し、デフォルトのグローバル ログ ファイル **/var/opt/perf/datafiles/logglob** が表示されます。デフォルトを使用しても、別のログ ファイルを指定してもかまいません。

**構文**

**logfile [***logfile***]**

**使い方**

ログ ファイルをオープンするには、生ログ ファイル名または抽出ログ ファイル名を指定します。export コマンドにより作成されるファイルの名前は指定できません。抽出ログ ファイル名を指定する場合、すべての情報がこのファイルから得られます。生ログ ファイル名を指定する場合、生ログ ファイルにアクセスするために、グローバル ログ ファイル名を指定する必要があります。この場合、指定するのは生ログ ファイル名のみです。

ログ ファイルが現在の作業ディレクトリにない場合、そのパスを入力する必要があります。

グローバル ログ ファイルとその他の生ログ ファイルは同じディレクトリにある必要があります。これらのファイル名は次のとおりです。

| | |
|---|---|
| `logglob` | グローバル ログ ファイル |
| `logappl` | アプリケーション ログ ファイル |
| `logproc` | プロセス ログ ファイル |
| `logdev` | |

| | |
|---|---|
| logtran | デバイス ログ ファイル |
| logindx | トランザクション ログ ファイル<br>索 引 ログ ファイル |

ログ ファイルをオープンすると、そのログ ファイルの目 次 が表 示 されます。

**メモ :** 生 ログ ファイル名 は変 更しないようにしてください。生 ログ ファイルにアクセスするときに、プログラムは標 準 ログ ファイル名 が有 効 であると仮 定します。複 数 のシステムのログ ファイルを同じシステムに配 置して分 析 を行うために、ログ ファイル名 を変 更 する必 要 がある場 合 、まずデータを抽 出してから抽 出 ログ ファイルの名 前 を変 更してください。

**例**

これは、典 型 的 なデフォルト グローバル ログ ファイルのリストです。

```
Global      file: /var/opt/perf/datafiles/logglob, version D

Application file: /var/opt/perf/datafiles/logappl

Process     file: /var/opt/perf/datafiles/logproc

Device      file: /var/opt/perf/datafiles/logdev

Transaction file: /var/opt/perf/datafiles/logdev

Index       file: /var/opt/perf/datafiles/logindx

System ID:  homer

System Type 9000/715/ S/N 2223334442 O/S HP-UX B.10.20 A

Data collector: SCOPE/UX C.02.30

File Created:  10/08/99

Data Covers:   44 days to 11/20/99

Shift is:      All Day

Data records available are:

Global Application Process Disk Volume Transaction

Maximum file sizes:

Global=10.0 Application=10.0 Process=20.0 Device=10.0 Transaction=10.0
MB

The first GLOBAL      record is on 10/08/99 at 08:17 AM

The first NETIF       record is on 10/08/99 at 08:17 AM

The first APPLICATION record is on 11/17/99 at 12:15 PM

The first PROCESS     record is on 10/08/99 at 08:17 AM

The first DEVICE      record is on 10/31/99 at 10:45 AM

The Transaction data file is empty
```

```
The default starting date & time = 10/08/99 11:50 AM   (LAST -30)
The default stopping date & time = 11/20/99 11:59 PM   (LAST -0)
```

# lvolume

lvolume コマンドは、抽出またはエクスポートする論理ボリューム データの種類を指定するときに使用します(このコマンドは、HP-UX システム上でのみ使用されます)。

デフォルトは、lvolume off です。

**構文**

| | |
|---|---|
| **lvolume** | **[on]**<br>**[detail]**<br>**[summary]**<br>**[both]**<br>**[off]** |

**パラメータ**

| | |
|---|---|
| on または detail<br>summary<br>both<br>off | この章の最初に説明されている「application」(137ページ) コマンドの説明にある「パラメータ」を参照してください。summary および both はエクスポートのみ可能です。 |

**例**

この例では、新しい抽出ログファイル **rx899** が作成され、同じ名前の既存のファイルが削除されます。8 月 1 日から 8 月 31 日までに収集されたすべての論理ボリューム データが抽出されます。

```
logfile /var/opt/perf/datafiles/logglob
output rx899,purge
start 08/01/99
stop 08/31/99
global detail
lvolume detail
month 9908
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -f rx899,purge -gz -xm 9908
```

# menu

menu コマンドは、使用可能な **extract** のコマンドのリストを出力するときに使用します。

**構文**

**menu**

**例**

| コマンド | パラメータ | 機能 |
|---|---|---|
| HELP | [topic] | コマンドとオプションの情報を取得 |
| GUIDE | 不慣れなユーザーのためにガイド コマンド モードを開始 | |
| LOGFILE | [logname] | 処理するログファイルを指定 |
| LIST | [filename\|*] | リスト ファイルを指定 |
| OUTPUT | [filename]<br>[,NEW/PURGE/APPEND] 出力先ファイルを指定 | |
| REPORT | [filename][,SHOW] | エクスポート フォーマットを指定 |
| GLOBAL | [DETAIL/SUMMARY/BOTH/OFF] | グローバルレコードを抽出 |
| APPLICATION | [DETAIL/SUMMARY/BOTH/OFF] | アプリケーション レコードを抽出 |
| PROCESS | [DETAIL/OFF/KILLED][APP=] | プロセスレコードを抽出 |
| DISK | [DETAIL/SUMMARY/BOTH/OFF] | ディスク デバイスレコードを抽出 |
| LVOLUME | [DETAIL/SUMMARY/BOTH/OFF] | 論理ボリューム レコードを抽出 |
| NETIF | [DETAIL/SUMMARY/BOTH/OFF] | 論理 NETIF レコードを抽出 |
| CPU | [DETAIL/SUMMARY/BOTH/OFF] | CPU レコードを抽出 |
| FILESYSTEM | [DETAIL/SUMMARY/BOTH/OFF] | ファイルシステム レコードを抽出 |

| | | |
|---|---|---|
| CONFIG | [DETAIL/OFF] | 設定レコードをエクスポート |
| CLASS classname | [DETAIL/SUMMARY/BOTH/OFF] | クラス名レコードをエクスポート |
| TRANSACTION | [DETAIL/SUMMARY/BOTH/OFF] | トランザクション レコードを抽出 |
| START | [startdate time] | スキャンの開始日時を指定 |
| STOP | [stopdate time] | スキャンの終了日時を指定 |
| SHIFT | [starttime - stoptime] [NOWEEKENDS] | シフト時間を指定 |
| SHOW | [ALL] | 現在のプログラム設定を表示 |
| EXPORT | [d/w/m/y][-offset] | ログファイル レコードを HOST フォーマット ファイルにコピー |
| EXTRACT | [d/w/m/y][-offset] | 選択したレコードを出力ファイルにコピー (または追加) |
| WEEKLY | [ww/yyww] | 1 週間のデータを自動ファイル名で抽出 |
| MONTHLY | [mm/yymm] | 1 か月のデータを自動ファイル名で抽出 |
| YEARLY | [yy/yyyy] | 1 年間のデータを自動ファイル名で抽出 |
| WEEKDAYS | [1...7] | export 1=Sunday ! または SH [command] から除外する日を設定 システム コマンドを実行 |
| MENU または ? | | コマンド メニュー (このリスト) を一覧表示 |
| EXIT または Q | | プログラムを終了 |

# monthly

monthly コマンドは、暦月に基づいてデータの抽出を指定するときに使用します。実行中、このコマンドは抽出するデータの月と年に基づいて、開始日と終了日を適切な日付に設定します。

出力ファイル名は、文字 **rxmo** の後に、抽出する年を示す4桁の数字と月を示す2桁の数字が続きます。たとえば、1999年 3月に抽出したデータは、**rxmo199903** ファイルに出力されます。

**構文**

| | |
|---|---|
| **monthly** | [*yymm*]<br>[*mm*] |

**パラメータ**

| | |
|---|---|
| `monthly` | 当月 (デフォルト) からデータを抽出します。 |
| monthly *mm* | 現在の年のデータから特定の月のデータを抽出します (*mm* は 01 ～ 12 の数字です)。 |
| monthly*yymm* | 特定の月と年のデータを抽出します (*yymm* は年の最後の2桁を示す数字と月を示す2桁の数字から構成されます)。たとえば、1999年 2月のデータを抽出するには、**monthly 9902** と指定します。 |

monthly コマンドを実行する前にログファイルを指定しない場合は、デフォルトの **logglob** ファイルが使用されます。

**使い方**

アーカイブするデータを月単位で抽出するときは、 monthly コマンドを使用します。

抽出して要約するデータ型は、`extract` コマンドの通常の規則に従い、monthly コマンドを実行する前に設定できます。月間出力ファイルがない場合、これらの設定値が適用されます。月間出力ファイルがある場合、データは最初に指定したデータ型に基づいてそのファイルに追加されます。

monthly コマンドには、前月の抽出ファイルをオープンし、そのファイルが完成しているか、つまりその月の最終日までの抽出データが含まれているかを確認する機能があります。ファイルが完成していない場合、monthly コマンドはデータをこのファイルに追加し、前月の抽出を完了します。

たとえば、monthly コマンドを 1999年 5月 7日に実行すると、5月 1日から現在の日付 (5月 7日) までのデータを含むログファイル **rxmo199905** が作成されます。

1999年 6月 4日に、再度 monthly コマンドを実行すると、当月の **rxmo199906** ファイルが作成される前に、前月の **rxmo199905** ファイルがオープンして確認されます。このファイルが完成していない場合は、データが追加され、1999年 5月 31日までの抽出を完了します。次に、**rxmo199906** ファイルが作成され、1999年 6月 1日から現在の日付 (6月 4日) までのデータを保持します。

少なくとも月 1回 monthly コマンドを実行すると、この機能が翌月のファイルを作成する前に、各月のファイルを完成します。**rxmo199905** (5月) と **rxmo199906** (6月) など、2つの連続した月間ファイルがあるときには、最初のファイルが対応する月に対して完成していると見なされるため、そのファイルをアーカイブして削除してもかまいません。

**メモ :**monthly コマンドとextract month コマンドは、どちらも暦月のデータを抽出する点で類似しています。monthly コマンドは、output コマンドの設定値を無視し、事前に定義されている出力ファイル名を代わりに使用します。また、このコマンドは、前月の抽出ログファイルがシステム上に残っている場合、欠落データをこのファイルに追加しようとします。一方、extract month コマンドは、output コマンドの設定値を使用します。このコマンドは、前月の抽出ファイル名を認識しないため、このファイルにデータを追加できません。

**例**

この例では、1999 年 5 月に記録された詳細アプリケーション データが抽出されます。

```
logfile /var/opt/perf/datafiles/logglob
global off
application detail
monthly 9905
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -a -xm 9905
```

# netif

netif コマンドは、**extract** または **export** を実行する論理ネットワーク インターフェイス (LAN) データの種類を指定するときに使用します。netif データは **logdev** ファイルに記録されます。

デフォルトは、netif off です。

**構文**

| | |
|---|---|
| **netif** | **[on]**<br>**[detail]**<br>**[summary]**<br>**[both]**<br>**[off]** |

**パラメータ**

| | |
|---|---|
| on または detail<br>summary<br>bothoff | この章の最初に説明されている「application」(137ページ) コマンドの説明にある「パラメータ」を参照してください。summary および both はエクスポートのみ可能です。 |

**例**

この例では、2000 年 3 月 1 日から 2000 年 6 月 30 日の平日午前 8 時から午後 5 時の間に収集された詳細 netif データが抽出されます。

```
logfile /var/opt/perf/datafiles/logglob

start 03/01/00

stop 06/30/00

shift 8:00 AM - 5:00 PM noweekends

netif detail

extract
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -n -b 03/01/00 -e 6/30/00 -s 8:00 am - 5:00 noweekends -xt
```

# output

output コマンドは、extract 関数または export 関数用の出力ファイル名を指定するときに使用します。

2 つ目のパラメータはオプションであり、同じ名前の出力ファイルが存在する場合の動作を指定します。

**構文**

| | | |
|---|---|---|
| **output** | **[** *filename***]** | **[,new]**<br>**[,purge]**<br><br>**[,append]** |

**パラメータ**

| | |
|---|---|
| ,new | 出力ファイルを新しいファイルにすることを指定します。これはバッチモードのデフォルトの動作です。同じ名前のファイルが存在する場合、バッチ ジョブが終了します。 |
| ,purge | 既存のファイルを削除して、新しい出力ファイル用に空きスペースを作ることを指定します。 |
| ,append | 既存の抽出ファイルにデータを追加することを指定します。指定した名前の出力ファイルが存在しない場合、新しいファイルが作成されます。 |

**使い方**

バッチ モードで動作を指定しない場合は、デフォルトの動作である ,new が使用されます。対話型モードでは、重複するファイルが見つかると動作の入力を促すプロンプトが表示されます。

出力ファイルを指定しない場合、デフォルトの出力ファイルが作成されます。デフォルトの出力ファイル名は次のとおりです。

**extract** の場合:**rxlog**

**export** の場合:

```
xfrdGLOBAL.ext
xfrsGLOBAL.ext
xfrdAPPLICATION.ext
xfrsAPPLICATION.ext
xfrdPROCESS.ext
xfrdDISK.ext
xfrsDISK.ext
xfrdLVOLUME.ext
xfrsLVOLUME.ext
xfrdNETIF.ext
xfrsNETIF.ext
xfrdCPU.ext
xfrsCPU.ext
xfrdFILESYSTEM.ext
xfrsFILESYSTEM.ext
xfrdTRANSACTION.ext
xfrsTRANSACTION.ext
xfrdCONFIGURATION.ext
```

ここで、**ext** は、**asc** (ASCII)、**dat** (データ ファイル)、**bin** (バイナリ)、**wk1** (表 計 算 シート) のいずれかです。

エクスポートの操 作 と共 に次 のように **stdout** (または *) という特 別 なファイル名 を使 用 すると、出 力 を直 接 **stdout** ファイルに送 信 できます (ただし、通 常 はシェル コマンドを使 用 して端 末 またはワークステーションをリダイレクトできます)。

**output stdout**

または

**output ***

出 力 をデフォルトの設 定 値 に返 すには、次 のように入 力 します。

**output default**

または

**output -**

**メモ :**エクスポート テンプレート ファイルで output パラメータを使 用 すると、エクスポート ファイルのデフォルトの出 力 ファイル名 を無 効 にできます。

**メモ :** 抽 出 操 作 のファイルは ASCII デバイスに対 応 していないため、**stdout** に出 力 できません。また、同 じ理 由 によりエクスポート操 作 のバイナリ フォーマット または WK1 フォーマットも **stdout** ファイルに出 力 できません。

**メモ :** 既 存 のエクスポート データ ファイルへの抽 出 データの追 加 や、既 存 の抽 出 ファイルへのエクスポート データの追 加 を行 わないようにしてください。間 違 ったデータ型 を追 加 しようとすると、エラーが生 じます。

**例**

この例では、出力ファイルが指定されていないため、抽出するアプリケーション要約データには、デフォルトの出力ファイル **rxlog** が使用されます。 ,purge オプションは、既存の出力ファイルを削除することを指定しています。

```
extract>
logfile /var/opt/perf/datafiles/logglob
output rxlog,purge
global off
application detail
extract month 9905
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -f rxlog,purge -a -xm 9905
```

# process

process コマンドは、プロセス データを抽出またはエクスポートするかどうかを指定するときに使用します。

デフォルトは、process off です。

**構文**

| | | |
|---|---|---|
| | **[on]** | |
| **process** | **[detail]** | **[application=#[-#] ,...]** |
| | **[off]** | |
| | **[killed]** | |
| | | |

**パラメータ**

| | |
|---|---|
| on | プロセス データの抽出またはエクスポートを指定します。 |
| detail | process detail の指定は、process on の指定と同じです。 |
| off | プロセス データを抽出またはエクスポートしないことを指定します。 |
| killed | killed を含む対象となる理由があるプロセスのみを指定します(測定インターバル中に終了したプロセス)。 |
| application | 選択したアプリケーションに属するプロセスのみを指定します。アプリケーションは 1 つの番号またはアプリケーション番号の範囲として入力できます (7 - 9 はアプリケーション 7、8、9 を意味します)。アプリケーション番号は、データ収集時に parm ファイルのアプリケーション定義の順序により決定されます。複数のアプリケーションを指定している場合、それぞれのアプリケーションをカンマで区切ります。 |

**メモ :** プロセス データにより、抽出ログ ファイルのサイズが大幅に増大する可能性があります。分析を行うためにワークステーションにログ ファイルをコピーする場合、抽出するプロセス データの量を制限できます。

**例**

この例では、process コマンドがインターバル中に終了したプロセスのうち、アプリケーション 1、4、6、7、8、10 に属するプロセスを指定します。**utility** プログラムの scan コマンドを使用して、特定のアプリケーションのアプリケーション番号を検索します。

```
process killed applications=1,4,6-8,10
```

# quit

quit コマンドは、**extract** プログラムを終了するときに使用します。quit コマンドは、 **extract** プログラムの exit コマンドに相当します。

**構文**

```
quit
```

```
q
```

# report

report コマンドは、export 関数で使用するエクスポート テンプレート ファイルを指定するときに使用します。エクスポート テンプレート ファイルが指定されていない場合は、デフォルトのエクスポート テンプレート ファイル **reptfile** が使用されます。エクスポート テンプレート ファイルは、export 関数で使用する出力フォーマットのさまざまな属性を指定するときに使用します。また、エクスポートするメトリックも指定します。

対話型モードでエクスポート テンプレート ファイルを指定しない場合、必要なデータ型のすべてのメトリックは ASCII フォーマットでエクスポートされます。

**構文**

**report [***exporttemplatefile***] [,show]**

**パラメータ**

| | |
|---|---|
| ,show | エクスポート テンプレート ファイルで指定されたすべてのメトリックについて、フィールドの位置と開始カラムをリストすることを指定します。このリストは、エクスポート ファイルを別のプログラムで処理するときに使用できます。 |

**使い方**

このコマンドを発行すると、事前に指定しているログ ファイルでエクスポート テンプレートのメトリックを検証するかしないかを確認するメッセージが表示され、入力を促すプロンプトが表示されます。この検証により、エクスポート テンプレート ファイルで指定されているメトリックがログ ファイルに存在することが確認されます。これにより、エクスポート テンプレート ファイルのエラーを確認できます。検証を実行しないと、エクスポートを実行するまでこの動作が延期されます。

**メモ :** ここで説明した report コマンドの ,show パラメータは、後述の show コマンドとは異なります。

# sh

sh は、**extract** を終了せずにシェルコマンドを入力するときに使用します。 **sh** またはエクスクラメーション マーク (**!**) の後に UNIX シェルコマンドを指定します。

**構文**

**sh** または **! [shell command]**

**パラメータ**

| | |
|---|---|
| sh ls | ls コマンドを実行し、**extract** に戻ります。このシェルコマンドは任意のシステムコマンドです。 |
| !ls | 上記と同じです。 |
| !ksh | Korn シェルを開始します。直ちに **extract** には戻りません。**extract** プログラムに戻るときは、**exit** と入力するか、**CTRL-d Return** を押します。 |

**使い方**

1 つのコマンドを実行すると、自動的に **extract** に戻ります。コマンドを実行するたびに **extract** に戻らないで、複数のシェルコマンドを発行する場合は、次のように新しいシェルを開始します。

シェルコマンドの名前を指定せずに sh コマンドを発行すると、次のようにシェルコマンドを指定するように促すプロンプトが表示されます。たとえば、次のようになります。

```
sh

enter SYSTEM command:ls
```

# shift

shift コマンドは、データの抽出を 1 日のうちの作業シフトに相当する時間や、週末 (土曜日および日曜日) 以外の日に限定するときに使用します。

デフォルトは shift all day で、週末を含む毎日のデータを抽出します。

**構文**

| | |
|---|---|
| **shift** | [*starttime-stoptime*]<br>**[all day]**<br>**[noweekends]** |

**パラメータ**

starttime パラメータとstoptime パラメータは、start コマンドの時間と同じフォーマットで入力します。午前 0 時をまたがるシフトも指定できます。starttime を stoptime より後の時刻にスケジュールすると、シフトは starttime に開始し、一晩中処理が行われ次の日の stoptime に終了します。

| | |
|---|---|
| `all day` | デフォルトのシフトである 12:00 AM - 12:00 AM (または 24 時間制で示す場合 00:00 - 00:00) を指定します。 |
| `noweekends` | 土曜日と日曜日に記録されたデータを除外することを指定します。午前 0 時をまたがるシフトと共に noweekends を入力すると、土曜日または日曜日に開始するシフトで週末が構成されます。 |

**例**

この例では、1999 年 6 月 15 日から毎日 10:00 AM から 4:00 PM までに収集されたディスク詳細データが抽出されます。

```
extract>
logfile /var/opt/perf/datafiles/logglob
global off
disk detail
shift 10:00 am - 4:00 PM
start 6/15/99
extract
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -d -b 6/15/99 -s 10:00 AM-4:00 PM -xt
```

# show

show コマンドは、オープンしたファイルの名前と設定可能な **extract** パラメータのステータスのリストを作成するときに使用します。

**構文**

```
show [all]
```

> **メモ :** ここで説明する show コマンドは、前述の report コマンドの ,show パラメータとは異なります。

**例**

show のみを使用すると、次のようなリストが作成されます。

```
ログファイル:   /var/opt/perf/datafiles/logglob
Output:     デフォルト
Report:     デフォルト
List:       "stdout"
The default starting date & time = 10/08/99 12:00 AM (LAST -30)
```

```
The default stopping date & time = 11/20/99 11:59 PM (LAST -0)

The default shift = 12:00 AM - 12:00 PM

GLOBAL           DETAIL           records will be processed

APPLICATION. . . . . . . . . . NO records will be processed

PROCESS . . . . . . . . . . . .  NO records will be processed

DISK DEVICE. . . . . . . . . . NO records will be processed

LVOLUME. . . . . . . . . . . . NO records will be processed

TRANSACTION. . . . . . . . . . NO records will be processed

NETIF . . . . . . . . . . . . .NO records will be processed

CPU . . . . . . . . . . . . . .NO records will be processed
FILESYSTEM. . . . . . . . . . .NO records will be processed

Configuration . . . . . . . . .NO records will be processed
```

show all を使用すると、次の例のようにより詳細なリストが作成されます。

```
ログファイル:          /var/opt/perf/datafiles/logglob

Global       file: /var/opt/perf/datafiles/logglob,version D

Application file: /var/opt/perf/datafiles/logappl

Process     file: /var/opt/perf/datafiles/logproc

Device      file: /var/opt/perf/datafiles/logdev

Transaction file: /var/opt/perf/datafiles/logdev

Index       file: /var/opt/perf/datafiles/logindx

System ID:  homer

System Type 9000/715/ S/N 2223334442 O/S HP-UX B.10.20 A

Data collector: SCOPE/UX C.02.30

File Created:   10/08/99

Data Covers:    44 days to 11/20/99

Shift is:       All Day

Data records available are:

  Global Application Process Disk Volume Transaction

Maximum file sizes:

  Global=10.0 Application=10.0 Process=20.0 Device=10.0

  Transaction=10.0 MB

Output:  Default

Report:  Default

List:    "stdout"
```

```
The default starting date & time = 10/08/99 11:50 AM (LAST -30)
The default stopping date & time = 11/20/99 11:59 PM(LAST - 0)
The default shift = 12:00 AM - 12:00 PM
GLOBAL...........DETAIL...........records will be processed
APPLICATION....................NO records will be processed
PROCESS........................NO records will be processed
DISK DEVICE....................NO records will be processed
LVOLUME........................NO records will be processed
TRANSACTION....................NO records will be processed
NETIF..........................NO records will be exported
CPU............................NO records will be processed
FILESYSTEM.....................NO records will be processed
Configuration .................NO records will be exported
Export Report Specifications:
   Interval = 3600, Separator = " "
   Missing data will not be displayed
   Headings will be displayed
   Date/time will be formatted
   Days to exclude: None
```

# start

start コマンドは、extract 関数と export 関数の開始日時を設定するときに使用します。デフォルトの開始日は、ログファイル内の最終日より 30 日前の日付です。また、30 日未満のレコードしかない場合、デフォルトの開始日はログファイル内の最初のレコードの日付になります。

**構文**

| | |
|---|---|
| **start** | **[date [*time*]]**<br>**[today [-*day*][*time*]]**<br>**[last [-*days*][*time*]]**<br>**[first [+*days*][*time*]]** |

**パラメータ**

| | |
|---|---|
| date | date フォーマットは、使用しているシステムに構成されている母国語によって異なります。母国語を使用しない場合やデフォルトの言語を C に設定している場合、データフォーマットは *mm/dd/yy* (月 / 日 / 年 ) になります。たとえば、extract 関数または export 関数の場合、1999 年 9 月 30 日は 09/30/99 になります。 |

| | |
|---|---|
| time | time フォーマットも、使用している母国語によって異なります。C 言語の場合、フォーマットは hh:mm AM または hh:mm PM (12 時間制の時 : 分の後に AM または PM を付ける形式 ) になります。たとえば、午前 7 時は 07:00 AM になります。<br>24 時間制の時刻もすべての言語で使用できます。たとえば、11:30 PM を 23:30 とすることもできます。<br>日付または時刻を不適切なフォーマットで入力すると、正しいフォーマットの例が表示されます。<br>開始時刻を指定しないと、午前 0 時 (12:00 AM) と見なされます。指定日の午前 0 時が開始時刻の場合、その日の始まり (24 時間制の場合 00:00) に開始されます。 |
| today | 現在の日付を指定します。today-days などのパラメータの修飾は、現在の日付からさかのぼる日数を指定します。たとえば、today-1 は前日を示し、today-2 は前前日を示します。 |
| last | ログファイルに含まれる最終日を表すときに使用します。last-days パラメータは、ログファイル内の最終日からさかのぼる日数を指定します。 |
| first | ログファイルに含まれる最初の日を表すときに使用します。first+days パラメータは、ログファイル内の最初の日から経過した日数を指定します。 |

**使い方**

次のコマンドは、start コマンドで設定されている開始日を無効にします。

- weekly
- monthly
- yearly
- extract (day、week、month、year のいずれかのパラメータを使用する場合 )
- export (day、week、month、 year のいずれかのパラメータを使用する場合 )

**例**

この例の start コマンドは、抽出する最初のインターバルの開始時刻として 1999 年 6 月 5 日 8:00 AM を指定します。output コマンドは出力ファイル **myout** を指定します。

```
logfile /var/opt/perf/datafiles/logglob

start 6/5/99 8:00 am

output myout

global detail

extract
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -g -b 06/05/99 8:00 AM -f myout -xt
```

# stop

stop コマンドは、extract 関数または export 関数の終了日時を指定するときに使用します。

デフォルトの終了日時は、ログファイルに記録されている最後の日時になります。

**構文**

| | |
|---|---|
| `start` | **[date [***time***]]**<br>**[today [-***day***][***time***]]**<br>**[last [-***days***][***time*** ]]**<br>**[first [+***days***][***time***]]** |

**パラメータ**

| | |
|---|---|
| `date` | date フォーマットは、使用しているシステムに構成されている母国語によって異なります。母国語を使用しない場合やデフォルトの言語を C に設定している場合、データフォーマットは *mm/dd/yy* (月 / 日 / 年 ) になります。たとえば、抽出関数またはエクスポート関数の場合、1999 年 9 月 30 日は 09/30/99 になります。 |
| `time` | time フォーマットも、使用している母国語によって異なります。C 言語の場合、フォーマットは *hh:mm* AM または *hh:mm* PM (12 時間制の時 : 分の後に AM または PM を付ける形式 ) になります。たとえば、午前 7 時は 07:00 AM になります。<br>24 時間制の時刻もすべての言語で使用できます。たとえば、11:30 PM を 23:30 とすることもできます。<br>日付または時刻を不適切なフォーマットで入力すると、正しいフォーマットの例が表示されます。<br>終了時刻を指定しないと、午前 0 時の 1 分前 (11:59 PM) と見なされます。指定日の午前 0 時 (12:00 AM) が終了時刻の場合、その日の終わり (24 時間制の場合 23:59) に終了されます。 |
| `today` | 現在の日付を指定します。today-days などのパラメータの修飾は、現在の日付からさかのぼる日数を指定します。たとえば、 today-1 は前日を示し、today-2 は前前日を示します。 |
| `last` | ログファイルに含まれる最終日を表すときに使用します。last-days パラメータは、ログファイル内の最終日からさかのぼる日数を指定します。 |
| `first` | ログファイルに含まれる最初の日を表すときに使用します。first+days パラメータは、ログファイル内の最初の日から経過した日数を指定します。 |

**使い方**

次のコマンドは、*stop* コマンドによって設定される終了日を無効にします。

- `weekly`
- `monthly`
- `yearly`
- extract (day、week、month、 year のいずれかのパラメータを使用する場合 )
- export (day、week、month、year のいずれかのパラメータを使用する場合 )

**例**

この例の stop コマンドは、抽出する最後のインターバルの終了時間として 1999 年 6 月 5 日 5:00 PM を指定します。 output コマンドは出力ファイル **myout** を指定します。

```
extract>
logfile /var/opt/perf/datafiles/logglob
start 6/5/99 8:00 AM
stop 6/5/99 5:00 PM
output myout
global detail
extract
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -g -b 6/5/99 8:00 AM -e 6/5/99 5:00 PM -f myout -xt
```

# transaction

transaction コマンドは、抽出またはエクスポートするトランザクション データの種類を指定するときに使用します。

**構文**

| | |
|---|---|
| **transaction** | **[on]**<br>**[detail]**<br>**[summary]**<br>**[both]**<br>**[off]** |

**パラメータ**

| | |
|---|---|
| on または detail<br>summary<br>both<br>off | この章の最初に説明されている「application」(137ページ) コマンドの説明にある「パラメータ」を参照してください。summary および both はエクスポートのみ可能です。 |

**例**

**rxmay99** という新しい抽出ログファイルが 1999 年 6 月 1 日に作成されます。この名前が付いている既存のファイルは削除されます。1999 年 5 月 1 日から 1999 年 5 月 31 日までに収集された生のトランザクション ログファイル データがすべて抽出されます。

```
extract>
logfile /var/opt/perf/datafiles/logglob
output rxmay99,purge
global detail
transaction detail
month 9905
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -gt -f rxmay99,purge -xm 9905
```

# weekdays

weekdays コマンドは、特定の曜日のデータをエクスポートから除くときに使用します (曜日 1 は日曜日を示します)。

**構文**

```
weekdays [1|2.....7]
```

**使い方**

1 週間のうち、特定の曜日のデータだけをエクスポートする場合は、このコマンドを使用してデータが不要な曜日を除きます。各曜日には、次の値が割り当てられています。

```
Sunday     =1
Monday     =2
Tuesday    =3
Wednesday  =4
Thursday   =5
Friday     =6
Saturday   =7
```

たとえば、月曜日から木曜日までに記録されたデータだけをエクスポートする場合は、金曜日、土曜日、日曜日のデータをエクスポートから除きます。

**例**

この例では、火曜日と木曜日に記録された詳細グローバル データがエクスポートから除外されます。出力エクスポート ファイルには、**myrept** エクスポート テンプレート ファイルで指定されているグローバル メトリックが含まれます。

```
extract>
logfile /var/opt/perf/datafiles/logglob
global detail
report myrept
weekdays 35
```

```
export
```

# weekly

weekly コマンドは、1 週間のデータ抽出を指定するときに使用します。1 週間は月曜日から日曜日までの 7 日間と定義されます。

実行中、このコマンドは抽出されたデータの週と年に基づき、開始日と終了日を適切な日付に設定します。

**構文**

| | |
|---|---|
| **weekly** | [*yyww*]<br>[*ww*] |

**パラメータ**

| | |
|---|---|
| `weekly` | 現在の週のデータを抽出します (デフォルト)。 |
| weekly*ww* | 現在の年のデータから特定の週のデータを抽出します (*ww* は、01 ～ 53 の任意の数字です)。 |
| weekly*yyww* | 特定の週と年のデータを抽出します ( *yyww* は、年の最後の 2 桁を示す数字と、年間を通した 2 桁の週数を示す数字から構成されます)。たとえば、1999 年 20 週目は、weekly 9920 になります。 |

weekly コマンドを実行する前にログファイルを指定しない場合は、**datafiles** ディレクトリのデフォルトの **logglob** ファイルが使用されます。

**使い方**

アーカイブするデータを週単位で抽出するときは、 weekly コマンドを使用します。

出力ファイル名は、文字 **rxwe** の後に、年の最後の 2 桁と抽出する週を示す 2 桁の週数が続きます。たとえば、1999 年の 12 週目 (3 月 22 日 月曜日 ～ 3 月 29 日 日曜日) は、**rxwe9912** という名前のファイルに出力されます。

抽出して要約するデータ型は、extract コマンドの通常の規則に従い、weekly コマンドを実行する前に設定できます。週間出力ファイルがない場合、これらの設定が適用されます。週間出力ファイルがある場合、最初に選択したデータの型に基づいてデータがそのファイルに追加されます。

weekly コマンドには、前の週の抽出ファイルをオープンし、そのファイルが完成しているか、つまりその週の最終日までの抽出データが含まれているかを確認する機能があります。ファイルが完成していない場合、weekly コマンドはデータをこのファイルに追加し、前の週の抽出を完了します。

たとえば、weekly コマンドを 1999 年 5 月 20 日 (木) に実行すると、5 月 17 日 (月) から現在の日付 (5 月 20 日) までのデータを含むログファイル **rxwe199920** が作成されます。

1999 年 5 月 26 日 (水) に、再度 weekly コマンドを実行すると、現在の週に対して **rxwe199921** ファイルが作成される前に、前の週の **rxwe199920** ファイルがオープンして確認されます。このファイルが完成していない場合は、データが追加され、1999 年 5 月 23 日 (日) までの抽出を完了します。次に、**rxwe199921** ファイルが作成され、1999 年 5 月 24 日 (月) から現在の日付 (5 月 26 日) までのデータを保持します。

少なくとも週 1 回 weekly コマンドを実行すると、この機能が翌週のファイルを作成する前に、各週のファイルを完成します。2 つの連続した週間ファイル (**rxwe199920** と **rxwe199921** など) があるときには、最初のファイルが対応する週に対して完成していると見なされるため、そのファイルをアーカイブして削除してもかまいません。

**メモ :** 週数は、その開始日に基づいて数えられます。したがって、年の最初の週 (週 01) は、その年の最初の月曜日から始まる週になります。その月曜日より前の日は、前年の最後の週に属します。weekly コマンドと extract week コマンドは、どちらも月曜日から日曜日までの 1 週間のデータを抽出する点で類似しています。weekly コマンドは、output コマンドの設定値を無視し、事前に定義されている出力ファイル名を代わりに使用します。また、このコマンドは、前の週の抽出ログファイルがシステム上に残っている場合、欠落データをこのファイルに追加しようとします。一方、extract week コマンドは、output コマンドの設定値を使用します。このコマンドは、前の週の抽出ファイル名を認識しないため、このファイルにデータを追加できません。出力ファイル名は、文字 **rxwe** の後に、現在の年 (*yyyy*) と週数 (*ww*) が続きます。

**例**

この例では、weekly コマンドにより現在の週のデータが抽出され、前の週の抽出ファイルがある場合は、このファイルが完成します。

```
extract>
logfile /var/opt/perf/datafiles/logglob
global detail
application detail
process detail
weekly
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -gap -xw
```

# yearly

yearly コマンドは、暦年に基づいてデータの抽出を指定するときに使用します。

実行中、このコマンドは抽出する年に基づいて、開始日と終了日を適切な日付に設定します。

**構文**

| | |
|---|---|
| **yearly** | **[***yyyy***]**<br>**[***yy***]** |

**パラメータ**

| | |
|---|---|
| yearly | 現在の年のデータを抽出します (デフォルト)。 |

| | |
|---|---|
| yearly *yy* | 特定の年のデータを抽出します (yy は、00 ～ 99 の数字です)。00 ～ 27 の指定は、2000 ～ 2027 年と見なされ、71 ～ 99 の指定は、1971 ～ 1999 年と見なされます。 |
| yearly *yyyy* | 特定の年のデータを抽出します (*yyyy* は、年を示す 1971 ～ 2027 の数字です)。 |

yearly コマンドを実行する前にログファイルを指定しない場合は、デフォルトの **logglob** ファイルが使用されます。

**使い方**

アーカイブするデータを年単位で抽出するときは、yearly コマンドを使用します。

出力ファイル名は、文字 **rxyr** の後に、抽出する年を示す 4 桁の数字が続きます。したがって、1999 年のデータは **rxyr1999** という名前のファイルに出力されます。

抽出して要約するデータ型は、extract コマンドの通常の規則に従い、 yearly コマンドを実行する前に設定できます。年間出力ファイルがない場合、これらの設定が適用されます。年間出力ファイルがある場合、データは最初に選択したデータ型に基づいてそのファイルに追加されます。

yearly コマンドには、前年の抽出ファイルをオープンし、そのファイルが完成しているか、つまりその年の最終日までの抽出データが含まれているかを確認する機能があります。ファイルが完成していない場合、yearly コマンドはデータをこのファイルに追加し、前年の抽出を完了します。

たとえば、 yearlyコマンドを 1998 年 12 月 15 日に実行すると、1998 年 1 月 1 日から現在の日付 (12 月 15 日) までのデータを含むログファイル **rxyr1998** が作成されます。

1999 年 1 月 5 日に、再度 yearly コマンドを実行すると、現在の年に対して **rxyr1999** ファイルが作成される前に、前年の **rxyr1998** ファイルがオープンして確認されます。このファイルが完成していない場合は、データが追加され、1998 年 12 月 31 日までの抽出を完了します。次に、 **rxyr1999** ファイルが作成され、1999 年 1 月 1 日から現在の日付 (1 月 5 日) までのデータを保持します。

少なくとも年 1 回 yearly コマンドを実行すると、この機能が翌年のファイルを作成する前に各年のファイルを完成します。2 つの連続した年間ファイル (**rxyr1998** と **rxyr1999** など) があるときには、最初のファイルが対応する年に対して完成していると見なされるため、そのファイルをアーカイブして削除してもかまいません。

前のパラグラフの説明は、生ログファイルが 1 年分のデータを保持できるサイズである場合にのみ当てはまります。生ログファイルのサイズを小さくして、より頻繁に (月 1 回など) yearly コマンドを実行する方が一般的です。

**メモ :**yearly コマンドと extract year コマンドは、どちらも暦年のデータを抽出する点で類似しています。yearly コマンドは、output コマンドの設定値を無視し、事前に定義されている出力ファイル名を代わりに使用します。また、このコマンドは、前年の抽出ログファイルがシステム上に残っている場合、欠落データをこのファイルに追加しようとします。一方、extract year コマンドは、output コマンドの設定値を使用します。このコマンドは、前年の抽出ファイル名を認識しないため、このファイルにデータを追加できません。

**例**

この例では、アプリケーション詳細データとグローバル要約データが既存の年間要約ファイルに追加されます (必要に応じて、このファイルが作成されます)。出力ファイルは **rxyr***yyyy* (*yyyy* は現在の年を表す) です。

```
extract>
logfile /var/opt/perf/datafiles/logglob
global detail
application detail
process off
yearly
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -ga -xy
```

# cpsh プログラムの使用

*cpsh プログラムを使用できるのは、HP Ops OS Inst to Realtime Inst LTU または Glance Pak Software LTU を有効にしている場合のみです。*

cpsh プログラムは、監視対象システムから収集されたリアルタイムのメトリック データを確認できる、新しいコマンド ライン プロンプトを表示します。

## インタラクティブ モードの使用

cpsh プログラムは、インタラクティブ モードで使用できます。オプションを指定せずに cpsh コマンドを実行すると、新しいコマンド プロンプトが表示されます。このプロンプトでは、リアルタイム メトリックの詳細を表示するためのさまざまなタスクを実行できます。

cpsh プロンプトを表示する手順は以下のとおりです。

1. HP Operations エージェントがインストールされているシステムに、ルートまたは管理権限でログオンします。
2. ローカル システムの cpsh プロンプトを表示するには、以下のコマンドを実行します。

   **cpsh**

   リモート システムの cpsh プロンプトを表示するには、以下のコマンドを実行します。

   **cpsh -n** *<システム名>*

   この *<システム名>* には、リモート システムの完全修飾ドメイン名を指定します。

   または

   **cpsh -n** *<IP アドレス>*

   この *<IP アドレス>* には、リモート システムの IP アドレスを指定します。

   > **メモ :**リモート システムの cpsh プロンプトを表示するときは、リモート システムで perfd プロセスが実行されている必要があります。cpsh ユーティリティによって、ローカル システムのパフォーマンス データに他のシステムがアクセスしないようにできます。詳細情報は、Restrict Access on page 56を参照してください。

   cpsh プロンプトが表示されます。

   見やすい形式でメトリック データを表示するには、-t オプションを指定して **cpsh** コマンドを実行します。

   たとえば、次のようになります。

   **cpsh -t**

   または

   **cpsh -n** <system_name> **-t**
3. cpsh プロンプトで使用できるコマンドの詳細を表示するには、**help** と入力します。

# リアルタイム メトリックの表 示

cpsh プロンプトを使 用することで、利 用 できるメトリックのリアルタイムの値 を表 示 できます。cpsh プロンプトで何らかの操 作 を実 行 するには、事 前 にメトリック コンテキストを設 定 する必 要 があります。perfd デーモンとその関 連 ユーティリティは、メトリック クラスに基 づいて利 用 可 能 データを処 理 します。このため、cpsh ユーティリティでリアルタイム データを表 示 する場 合 は、利 用 可 能 データの表 示 操 作 を実 行 する前 に、常 にメトリック クラスを設 定 する必 要 があります。

メトリック クラスのメトリックのリアルタイム値 を表 示 する手 順 は以 下 のとおりです。

1. cpsh プロンプトに **class** <*メトリック クラス*> と入 力 します。
2. そのクラスに現 在 設 定 されているメトリックのリストを表 示 するには、**list** と入 力 します。指 定 したメトリック クラスのすべてのデフォルト メトリックが表 示 されます。
3. 指 定 したクラスに属 すメトリックの値 を表 示 するには、cpsh プロンプトに **push** と入 力 します。cpsh プログラムは、該 当 メトリックのリアルタイム値 を表 形 式 で表 示 します。
4. cpsh プロンプトに戻 るには、**Ctrl** キーを押 しながら **C** キーを押 します。

# メトリック クラスの変 更

メトリック クラスのデフォルト メトリックのリストには、別 の使 用 可 能 メトリックを追 加 できます。cpsh プロンプトでメトリック クラスにメトリックを追 加 したり、削 除 する手 順 は以 下 のとおりです。

1. cpsh プロンプトを表 示 します。
2. cpsh プロンプトに**class** <*メトリック クラス*> と入 力 します。
3. **list** と入 力 します。指 定 したメトリック クラスのすべてのデフォルト メトリックが表 示 されます。
4. メトリックを削 除 する手 順 は以 下 のとおりです。

   cpsh プロンプトに **delete**<*メトリック名* > と入 力 します。

   **list** と入 力 します。指 定 したメトリック クラスのメトリック リストには、削 除 したメトリックは表 示 されなくなります。
5. メトリック クラスにメトリックを追 加 する手 順 は以 下 のとおりです。

   cpsh プロンプトに **add**<*メトリック名* > と入 力 します。

   **list** と入 力 します。指 定 したメトリック クラスのメトリック リストには、新 たに追 加 したメトリックが表 示 されます。

## すべての使 用 可 能 メトリックの表 示

メトリック クラスに属 すすべての使 用 可 能 メトリックを表 示 する手 順 は以 下 のとおりです。

1. cpsh プロンプトを表 示 します。
2. cpsh プロンプトに**class** <*メトリック クラス*> と入 力 します。
3. **list all** と入 力 します。指 定 したメトリック クラスに属 すすべての使 用 可 能 メトリックが表 示 されます。

## メトリック クラスの整理

メトリック クラスに対してメトリックの追加/削除処理を繰り返し実行する代わりに、メトリック クラスをまとめて整理することができます。メトリック クラスを整理して指定のメトリックを追加する手順は以下のとおりです。

1. cpsh プロンプトを表示します。
2. cpsh プロンプトに**class** <*メトリック クラス*> と入力します。
3. **init** <*メトリック名*> <*メトリック名*> <*メトリック名*> ... と入力します。

   指定したメトリック クラスには、init で指定したメトリックのみが含まれるようになります。

## メトリックのヘルプの表示

cpsh プロンプトでは、すべてのリアルタイム メトリックの説明を表示できます。メトリックの説明を表示する手順は以下のとおりです。

1. cpsh プロンプトを表示します。
2. class <*メトリック クラス*> と入力します。
3. cpsh プロンプトに **help** <*メトリック名*> と入力します。指定したメトリックの詳細な説明が表示されます。

## 要約メトリック データの表示

GLOBAL クラスと TABLE クラスのメトリックについては、cpsh プロンプトで要約データを表示できます。要約データを表示する手順は以下のとおりです。

1. cpsh プロンプトを表示します。
2. cpsh プロンプトに **class gbl** または **class tbl** と入力します。
3. **summ** <*間隔*> と入力します。この <*間隔*> には、要約の間隔を秒単位で指定します。<*間隔*> は、cpsh が接続されている perfd サーバーの収集間隔の倍数である必要があります。

   cpsh ユーティリティは、指定したメトリック クラスに属すメトリックの以下の測定値を表示します。

   Maximum (最大)

   Minimum (最小)

   Average (平均)

   Standard deviation (標準偏差)

# パフォーマンス アラーム

アラームを定義するときには、Performance Collection Component を使用します。アラームは、scope または DSI のメトリックが、定義しておいた条件を満たしたときや超えたときにユーザーに通知します。

アラームを定義するには、アラートまたは動作をトリガする条件を各監視対象システムに指定します。アラームは、アラーム定義ファイルである alarmdef で定義します。

データが scope または他のコレクタにより記録されると、そのデータはアラーム定義と比較され、条件を満たしているかどうかが確認されます。条件を満たしている場合は、アラートまたは動作がトリガされます。

リアルタイム アラーム ジェネレータを使用すると、以下のタスクを実行できます。

- HPOM コンソールへのアラート通知の送信
- アラーム通知が生成された際に、SNMP トラップを作成
- SNMP トラップを SNMP トラップ リスナーに転送
- 監視対象システム上のローカル アクションを実行

履歴ログ ファイル データをアラーム定義と比較して分析し、その結果を utility プログラムの analyze コマンドを使用してレポートできます。

## アラームの処理

Performance Collection Component により収集されたパフォーマンス データは、alarmdef ファイルで定義されたアラーム条件と比較され、その条件を満たしているかどうかが確認されます。条件を満たしている場合、アラームが発生し、アラームに対して定義された動作 (ALERT、PRINT、EXEC) が実行されます。

ログ ファイルに収集されたデータは、alarmdef ファイルのアラーム定義と比較されます。アラーム条件が満たされると、アラーム定義で定義された動作が実行されます。ただし、データがログ ファイルに記録されない場合 (たとえば、しきい値 パラメータが高位置に設定されている場合) は、alarmdef ファイルのアラーム条件が満たされても、アラームは生成されません。それぞれのメトリックのクラスのしきい値については、Thresholdsを参照してください。

アラーム定義には、以下のアクションを含めることができます。

- オペレーティング システム コマンドを使用して実行されるローカル アクション
- Network Node Manager (NNM) または HPOM に送信されるメッセージ

## アラーム ジェネレータ

Performance Collection Component アラーム ジェネレータは、アラーム通知の通信を処理します。アラーム ジェネレータは、アラーム ジェネレータ サーバー (perfalarm)、アラーム ジェネレータ データベース (agdb)、utility プログラム (agsysdb) で構成されます。

agdb には、SNMP ノードのリストが含まれます。agsysdb プログラムは、アラーム イベントにより実行される動作を表示するときや変更するときに使用します。

Performance Collection Component を起動すると、perfalarm が開始され、agdb を読み込み、アラーム通知の送信の有無およびその送信先を決定します。

アラート通知の送信先のリストを表示するには、次のコマンド ライン オプションを使用します。

```
agsysdb -l
```

## Network Node Manager への SNMP トラップの送信

SNMP トラップを Network Node Manager に送信するには、次のコマンドを使用して Performance Collection Component の agdb にシステム名を追加する必要があります。

agsysdb -add*systemname*

ALERT が生成されると、定義したシステムに SNMP トラップが送信されます。トラップ テキストには、ALERT と同じメッセージが含まれます。

システムへの SNMP トラップの送信を停止するには、次のコマンドを使用して agdb からシステム名を削除する必要があります。

agsysdb -delete*systemname*

## HPOM へのメッセージの送信

HPOM にアラート通知を送信することができます。デフォルトでは、アラーム ジェネレータは EXEC 文のアラームに定義されているローカル アクションを実行しません。代わりに、アラーム ジェネレータは HPOM のイベント ブラウザにメッセージを送信します。

次のコマンドを使用すると、HPOM への情報送信を停止するようにデフォルトを変更できます。

```
agsysdb -ovo OFF
```

Performance Collection Component トラップを他のノードに送信するには、/etc/services ファイルに以下のエントリを追加します。

```
snmp-trap   162/tcp  # SNMPTRAP

snmp-trap   162/udp  # SNMPTRAP
```

この例では、162 がポート番号を指定しています。Performance Collection Component からトラップを別のノードに送信する場合は、/etc/services ファイルの snmp-trap 文字列がチェックされます。このエントリがないと、トラップは別のノードに送信されません。

## ローカル アクションの実行

デフォルトでは、Performance Collection Componentは EXEC 文で指定されているローカル コマンドを実行しません。

次のようにデフォルトを変更すると、ローカル アクションを有効にできます。

```
agsysdb -actions always
```

次の表は、HP Operations Manager (HPOM) への情報の送信とローカル アクションの実行に関する設定をリストしたものです。

**表 9:HPOM への情報の送信とローカル アクションの実行に関する設定**

| フラグ | 運用監視コンポーネントが動作している場合 | 運用監視コンポーネントが動作していない場合 |
|---|---|---|
| **HPOM フラグ** | | |
| off | アラート通知がHPOMに送信されません。 | アラート通知がHPOMに送信されません。 |
| on | アラート通知がHPOMに送信されます。 | アラート通知がHPOMに送信されません。 |
| **ローカル アクション フラグ** | | |
| off | ローカルアクションが実行されません。 | ローカルアクションが実行されません。 |
| always | 運用監視コンポーネントが動作していても、ローカルアクションが実行されます。 | ローカルアクションが実行されます。 |
| on | ローカルアクションがHPOMに送信されます。 | ローカルアクションが実行されます。 |

## アラーム処理のエラー

アラーム送信時に起きた最後のエラーは、agdb に記録されます。agdb の内容を参照するには、次のように入力します。

```
agsysdb -l
```

次の情報が表示されます。

```
PA alarming status:

OVO messages : on      Last Error : none

Exec Actions : on

Analysis system: <hostname>, Key=<ip address>

PerfView : no Last Error : <error number>

SNMP : yes Last Error : <error number>
```

# アラームの履歴データの分析

ログファイルデータ内のアラーム状態を検索するには、utility プログラムの analyze コマンドを使用します (5 章「ユーティリティコマンド」を参照してください)。ログファイル内の履歴データを alarmdef ファイル内のアラーム定義と比較して、どのアラーム状態がトリガされているかを調べるため、この処理は前述のリアルタイムアラームの処理とは異なります。

## 履歴データのアラーム情報の例

次の例は、履歴データのアラーム状態を分析するときにレポートされる内容を示しています。

最初の例では、START 文、END 文、REPEAT 文がアラーム定義で定義されています。アラーム開始イベントは、指定の継続時間中、アラームがそのすべての条件を満たすたびにリストされます。これらの条件が満たされなくなると、アラーム終了イベントがリストされます。最初のアラームが終了する前に別のアラームが発生するほど長い期間アラーム条件が満たされている場合、繰り返しイベントがリストされます。

リストされる各イベントは、日時、アラーム番号、アラームイベントを示します。EXEC の動作は実行されませんが、要求されたすべてのパラメータ置換と共に所定の位置にリストされます。

```
05/10/99 11:15  ALARM [1] START
CRITICAL:CPU test 99.97%

05/10/99 11:20  ALARM [1] REPEAT
WARNING:CPU test 99.997%

05/10/99 11:25  ALARM [1] END
RESET:CPU test 22.86%
EXEC:end.script
```

カラーワークステーションを使用している場合、次の出力が強調表示されます。

CRITICAL 文は赤色で表示されます。

MAJOR 文はマゼンダで表示されます。

MINOR 文は黄色で表示されます。

WARNING 文はシアン（緑がかった青）で表示されます。

NORMAL 文は緑で表示されます。

次の例は、アラームイベントがリストされた後に表示されるアラームの要約を示しています。最初のカラムにはアラーム番号、2 番目のカラムにはアラーム状態が起きた回数、3 番目のカラムにはアラーム状態の全継続時間がリストされます。

```
Performance Alarm Summary:

          Alarm   Count   Minutes
              1     574      2865
              2       0         0

Analysis coverage using "alarmdef":

Start: 05/04/99 08:00    Stop: 05/06/99 23:59

Total time analyzed:  Days: 2  Hours: 15  Minutes: 59
```

# アラーム定義の構成要素

定義した1つ以上の条件が指定した継続時間以上続くと、アラームが発生します。アラーム定義には、アラームの開始時または終了時に実行する動作を含めることができます。

条件とは、複数の項目間での比較のことです。比較される項目は、メトリック名、定数、変数のいずれかです。たとえば、次のようになります。

```
ALARM gbl_cpu_total_util > 95 FOR 5 MINUTES
```

アラームの開始時、終了時、繰り返し時に動作を実行するように指定できます。次のいずれかの動作を指定できます。

- HPOM にメッセージを送信するか、NNM に SNMP トラップを送信する ALERT
- オペレーティング システム コマンドを実行する EXEC
- `utility` プログラムを使用して処理したときに、`stdout` にメッセージを送信する PRINT

たとえば、次のようになります。

```
ALARM gbl_swap_space_util > 95 FOR 5 MINUTES
   START
      RED ALERT "Global swap space is nearly full"
   END
      RESET ALERT "End of global swap space full condition"
```

ブール論理、アプリケーションなどの複数インスタンス データによるループ、変数を使用すると、より複雑な動作を生成できます(詳細は、「アラーム構文のリファレンス」の項のを参照してください)。

また、INCLUDE 文を使用すると、使用するその他のアラーム定義ファイルを識別できます。アラーム定義を小さいファイルに分割する場合などに、この操作を行います。

# アラーム構文のリファレンス

この項では、アラーム構文で使用できる文について説明します。構文を使用して有用なアラーム定義を作成する方法の例については、`alarmdef` ファイルを参照してください。

## アラーム構文

```
ALARM condition [[AND,OR]condition]
  FOR duration [SECONDS, MINUTES]

  [TYPE="string"]
  [SERVICE="string"]
  [SEVERITY=integer]
  [START action]
  [REPEAT EVERY duration [SECONDS, MINUTES] action]
  [END action]

[RED, CRITICAL, ORANGE, MAJOR, YELLOW, MINOR, CYAN, WARNING,
  GREEN, NORMAL, RESET] ALERT message

EXEC "UNIX command"
```

```
PRINT message
IF condition
   THEN action
   [ELSE action]

{APPLICATION, PROCESS, DISK, LVOLUME, TRANSACTION, NETIF, CPU,
   FILESYSTEM} LOOP action

INCLUDE "filename"

USE "data source name"

[VAR] name =  value

ALIAS name = "replaced-name"

SYMPTOM variable [TYPE = {CPU, DISK, MEMORY, NETWORK}]
  RULE condition  PROB probability
  [RULE condition PROB probability]
    .
    .
```

## 表記

- 中かっこ ({}) は、項目の1つを選択することを示します。
- 大かっこ ([ ]) は、オプションの項目を示します。
- 大かっこまたは中かっこ内でカンマで区切られている項目はオプションです。1つだけ選択してください。
- 斜体は、置換する変数名を示します。
- 構文キーワードは、すべて大文字です。

## 共通要素

次に説明する要素はアラーム構文のいくつかの文で使用されます。

- コメント
- 複合文
- 条件
- 定数
- 式
- メトリック名
- メッセージ

## コメント

コメントの前には、二重スラッシュ (//) またはポンド記号 (#) を付けます。どちらの場合も、コメントはその行の終わりで終了します。たとえば、次のようになります。

# any text or characters

または

// any text or characters

## 複合文

複合文により、文のリストを単一文として実行できます。複合文とは、中かっこ ({ }) 内の文のリストのことです。複合文は、IF 文や LOOP 文、ALARM 文の START、REPEAT、END の各節と共に使用します。複合文には、ALARM 文および SYMPTOM 文は含められません。

```
{
 statement
 statement
}
```

次の例では、highest_cpu により、変数が定義されています。highest_cpu 値は保存され、さらに大きな highest_cpu 値がこの highest_cpu 値を超えたときにのみユーザーに通知します。

```
highest_cpu = highest_cpu
IF  gbl_cpu_total_util > highest_cpu  THEN
  // Begin compound statement
  {
  highest_cpu = gbl_cpu_total_util
  ALERT "Our new high CPU value is ", highest_cpu, "%"
  }
  // End compound statement
```

## 条件

条件は、2 つの項目間の比較として定義されます。

```
item1 {>, <, >=, <=, ==, !=}item2
          [AND, OR[item3 {>, <, >=, <=, ==, !=}item4]]
```

ここで、「==」は「等号」を、「!=」は「非等号」を意味します。

条件は、ALARM 文、IF 文、SYMPTOM 文で使用されます。項目は、メトリック名、数値定数、引用符で囲まれた英数字の文字列、別名、変数のいずれかです。英数字の項目を比較するときは、== または != のみを演算子として使用できます。

## 定数

定数は、数値または英数字です。英数字の定数は、二重引用符で囲む必要があります。たとえば、次のようになります。

```
345
345.2
"Time is"
```

定数は、式や条件で有用です。たとえば、次のようにメトリックが高すぎるときにアラームを発生させるには、そのメトリックを条件内の定数値と比較します。

```
gbl_cpu_total_util > 95
```

## 式

演算式は、複数のオペランドに関して1つ以上の算術演算を実行します。数値を使用する場合、いつでも式を使用できます。適切な算術演算子は、次のとおりです。

```
+, -, *, /
```

かっこを使用すると、最初に評価する式の部分を制御できます。

たとえば、次のようになります。

```
Iteration + 1
gbl_cpu_total_util - gbl_cpu_user_mode_util
( 100 - gbl_cpu_total_util ) / 100.0
```

## メトリック名

アラーム定義でメトリック名を指定する場合、メトリックの現在の値が置き換えられます。メトリック名は、メトリックの定義に示されるとおりに正確に入力する必要があります。ただし、大文字と小文字を区別する必要はありません。メトリックの定義については、『Performance Collection Component Dictionary of Operating Systems Performance Metrics』を参照してください。

メトリックが、SCOPE データソース以外のデータソースのものである場合 (DSI メトリックなど) は、メトリックの完全修飾名を使用することをお勧めします。

メトリックの完全修飾名を指定するためのフォーマットは、次のとおりです。

```
data_source:instance(class):metric_name
```

SCOPE データソース内のグローバルメトリックは、修飾が不要です。たとえば、次のようになります。

```
metric_1
```

SCOPE データソースで定義されている各アプリケーションに使用可能なアプリケーション メトリックには、アプリケーション名が必要です。たとえば、次のようになります。

```
application_1:metric_1
```

`application`、`process`、`disk`、`netif`、`transaction`、`lvolume`、`cpu`、`filesystem` などの種類の複数インスタンス データの場合、LOOP 文を使用している場合を除いて、メトリックをデータ

型と関連付ける必要があります。この方法は、データ型名を指定し、その後にコロンを入力し、続いてメトリック名を入力します。たとえば、other_apps:app_cpu_total_util は、アプリケーション other_apps の合計 CPU 使用率を指定します。

> **注:** メトリックの完全修飾インスタンスを複数指定し、別名のインスタンス内で使用するときに、別名の 1 つにクラス識別名がある場合、次の例で示される構文を使用することをお勧めします。
>
> ```
> alias my_fs="/dev/vg01/lvol1(LVOLUME)"alarm my_fs:LV_SPACE_UTIL >
> 50 for 5 minutes
> ```

空白を含むアプリケーション名を使用する場合、空白をアンダースコア (_) に置換する必要があります。たとえば、application 1 は application_1 に変更する必要があります。特殊文字を含む名前や、大文字小文字に意味がある名前の使い方の詳細は、「ALIAS 文」を参照してください。

ディスク「other」とアプリケーション「other」がある場合、インスタンスと共にクラスも指定する必要があります。

```
other (disk):metric_1
```

抽出ログファイル内のグローバルメトリックは、次のように指定します (ここで scope_extract はデータソース名です)。

```
scope_extract:application_1:metric_1
```

DSI メトリックは、次のように指定します。

```
dsi_data_source:dsi_class:metric_name
```

> **注:** 特殊文字 (アスタリスクなど) を含むメトリック名はいずれも、指定する前に別名を指定する必要があります。

## メッセージ

メッセージとは、PRINT 文または ALERT 文によって送信される情報です。メッセージは、引用符で囲まれた英数字の文字列、数値定数、式、変数の任意の組み合わせで構成されます。メッセージ内の要素は、カンマによって区別されます。たとえば、次のようになります。

```
RED ALERT "cpu utilization=", gbl_cpu_total_util
```

数値定数、メトリック、式では、幅 (全桁数) と小数の桁数を設定できます。*width* はフォーマットするフィールドの幅を指定します。*decimals* は使用する小数の桁数を指定します。数値は右揃えです。- (マイナス記号) は左揃えを指定します。英数文字列は必ず左揃えです。たとえば、次のようになります。

```
metric names [|[-]width[|decimals]]

gbl_cpu_total_util|6|2   formats as '100.00'
(100.32 + 20)|6          formats as '   120'
gbl_cpu_total_util|-6|0  formats as '100   '
gbl_cpu_total_util|10|2  formats as '    99.13'
gbl_cpu_total_util|10|4  formats as '  99.1300'
```

# ALARM 文

ALARM 文は、1 つの条件または条件一式と、その条件が真になるための継続時間を定義します。ALARM 文では、アラーム状態の開始時、繰り返し時、終了時に実行される動作を定義できます。アラームとして定義できる条件またはイベントには、次のものがあります。

- グローバル スワップ スペースが、5 分間ほとんどいっぱいになっている場合
- メモリ ページング率が、1 インターバルについて高すぎる場合
- CPU が最後の 10 分間に 75% の利用率で動作している場合

## 構文

```
ALARM condition [[AND,OR]condition]
  FOR duration{SECONDS, MINUTES}
  [TYPE="string"]
  [SERVICE="string"]
  [SEVERITY=integer]
  [START action]
  [REPEAT EVERY duration {SECONDS, MINUTES} action]
  [END action]
```

- `ALARM` 文は、トップレベルの文である必要があります。ALARM 文は別の文にネストできません。ただし、いくつかの ALARM 条件を単一の ALARM 文に含めることはできます。複数の条件が `AND` でリンクされている場合、アラームをトリガするには、すべての条件が真になる必要があります。複数の条件が `OR` でリンクされている場合、いずれか 1 つの条件によってアラームがトリガされます。
- `TYPE` は、引用符で囲まれた 38 文字以内の文字列です。アラームを送信している場合は、`TYPE` を使用するとアラームを分類したり、使用するグラフ テンプレートの名前を指定できます。
- `SERVICE` は、引用符で囲まれた 200 文字以内の文字列です。ServiceNavigator を使用している場合、ServiceNavigator で定義したサービスと Performance Collection Component のアラームをリンクできます (『HP Operations ServiceNavigator Concepts and Configuration Guide』を参照してください)。

  ```
  SERVICE="Service_id"
  ```

- `SEVERITY` は、0 から 32767 までの整数です。
- `START`、`REPEAT`、`END` は、アラーム条件が満たされたとき、再び満たされたとき、終了したときに起きる動作を指定するために使用するキーワードです。ALARM 文では、`START`、`REPEAT`、`END` のうち少なくとも 1 つを必ず使用します。これらの各キーワードの後には、*action* が続きます。
- *action* — ALARM START、REPEAT、END のいずれかと共に最も頻繁に使用する動作は、ALERT 文です。EXEC 文を使用すると、メッセージを送信したりバッチ ファイルを実行できます。また、`utility` プログラムを使用して履歴ログ ファイルを分析している場合は、PRINT 文も使用できます。別の ALARM を除き、いずれの構文も正しい構文です。

  START、REPEAT、END の各動作は、複合文になります。たとえば、複合文を使用すると、ALERT と EXEC の両方を提供できます。

- *Conditions* – 条件は、2 つの項目間の比較として定義されます。

```
item1 {>, <, >=, <=, ==, !=}item2
   [AND, OR[item3 {>, <, >=, <=, ==, !=}item4]]
```

ここで、「==」は「等号」を、「!=」は「非等号」を意味します。

項目は、メトリック名、数値定数、引用符で囲まれた英数字の文字列、別名、変数のいずれかです。英数字の項目を比較するときは、== または != のみを演算子として使用できます。

二次条件間で演算子の「OR」と「AND」を指定して、複合条件を使用できます。たとえば、次のようになります。

```
ALARM gbl_cpu_total_util > 90 AND
gbl_pri_queue > 1 for 5 minutes
```

- 二次条件間で演算子の「OR」と「AND」を指定して、複合条件を使用できます。たとえば、次のようになります。

```
ALARM gbl_cpu_total_util > 90
gbl_cpu_sys_mode_util > 50 for 5 minutes
```

これにより、両方の条件が真になるとアラームが発生します。

FOR *duration* SECONDS, MINUTES は、アラームをトリガするために、条件が真のままでなければならない期間を指定します。

1分未満の継続時間を指定する場合、特に複数のデータソースがシステム上にあるときは注意してください。各データソースのデータを非常に短いインターバルでポーリングする必要がある場合は、パフォーマンスが著しく低下します。継続時間は、アラーム条件で示されているメトリックの最長収集インターバルの倍数にする必要があります。

収集インターバルより短い継続時間を設定しないでください。次の収集サイクルが完了するまでメトリックの値は変更されないため、収集インターバルより短い継続時間を設定すると、重複するアラームが不必要に処理されることになります。

parm ファイルで、collectioninterval パラメータにある次のパラメータでは、収集インターバルを制御します。

global:このパラメータは、scope および dsi ログファイルに記録される、プロセスメトリックを除くすべてのメトリックの収集インターバルを秒単位で示します。デフォルトでは5分に設定されています。

process:このパラメータは、プロセスメトリックの収集インターバルを秒単位で示します。

collectioninterval パラメータの詳細については、「データ記録間隔の設定」を参照してください。

- REPEAT EVERY*duration*SECONDS, MINUTES は、アラームが繰り返される前の期間を指定します。

## 使い方

アラームサイクルは、AND でリンクされたすべてのアラーム条件、または OR でリンクされたアラーム条件の1つが、少なくとも指定の継続時間、真であった最初のインターバルに開始します。その時間に、アラームジェネレータは、START *action* を実行し、その後の各インターバルで REPEAT 条件を確認します。十分な時間が確認されると、REPEAT 節の *action* が実行されます(この処理は1つ以

上 のアラーム条 件 が偽 になるまで続きます)。この処 理 がアラーム サイクルを完 了 し、END statement がある場 合 、それが実 行 されます。

アラームを通 知 するには、START 文 および END 文 で ALERT 文 を使 用 します。END ALERT を指 定 しないと、アラーム ジェネレータは自 動 的 にアラームを HPOM に送 信 し、SNMP トラップを NNM に送 信 します。

## 例

次 の ALARM の例 では、スワップ利 用 率 が 5 分 間 高 い場 合 に、レッド アラートを送 信 しています。これは、デフォルトの `alarmdef` ファイルのアラーム状 態 と類 似 しています。この例 を `alarmdef` ファイルに追 加 する場 合 は、必 ずデフォルトのアラーム状 態 を削 除 してください。削 除 しないと、その後 のアラート メッセージが混 乱 する場 合 があります。

```
ALARM  gbl_swap_space_util > 90 FOR 5 MINUTES
    START
      RED ALERT "swap utilization is very high "
   REPEAT EVERY 15 MINUTES
      RED ALERT "swap utilization is still very high "
    END
      RESET ALERT "End of swap utilization condition"
```

この ALARM の例 では、`gbl_swap_space_util` メトリックをテストして、それが 90 より大 きいかどうかを確 認 しています。アラーム ジェネレータの構 成 方 法 により、ALERT を SNMP トラップを介 して NNM に送 信 したり、メッセージとして Operations Manager に送 信 できます。

REPEAT 文 は、15 分 ごとに `gbl_swap_space_util` 条 件 を確 認 します。メトリックが 90 より大 きい間 は、REPEAT 文 が 15 分 ごとにメッセージ「`swap utilization is still very high`」を送 信 します。

`gbl_swap_space_util` 条 件 が 90 以 下 になると、RESET ALERT 文 がメッセージ「`End of swap utilization condition`」と共 に送 信 されます。

次 の例 では、ALARM 文 での複 合 動 作 を定 義 しています。この例 は、イベント発 生 時 にメッセージを送 信 する方 法 を示 しています。

```
ALARM gbl_cpu_total_util > 90 FOR 5 MINUTES
  START
  {
    RED ALERT "Your CPU is busy."
    EXEC "echo 'cpu is too high'| mailx root"
  }
  END
     RESET ALERT "CPU no longer busy."
```

ALERT は、NNM への SNMP トラップの送 信 または HPOM へのメッセージの送 信 をトリガできます。EXEC は、アラーム ジェネレータの構 成 方 法 により、ノード 上 のローカル アクションとしてメール メッセージの送 信 をトリガできます。

次 の 2 つの例 は、複 数 の条 件 の使 い方 を示 したものです。ALARM 文 には、複 数 のテスト条 件 を指 定 できます。この場 合 、送 信 される ALERT に対 してそれぞれの文 が真 である必 要 があります。

次の ALARM の例では、gbl_cpu_total_util メトリックと gbl_cpu_sys_mode_util メトリックをテストしています。両方の条件が真である場合、RED ALERT 文がレッド アラートを送信します。テスト条件のいずれかが偽になると、RESET が送信されます。

```
ALARM gbl_cpu_total_util > 90
  AND gbl_cpu_sys_mode_util > 50 FOR 5 MINUTES
START
  RED ALERT "CPU busy and Sys Mode CPU util is high."
END
  RESET ALERT "The CPU alert is now over."
```

次の ALARM の例では、gbl_cpu_total_util メトリックと gbl_cpu_sys_mode_util メトリックをテストしています。いずれかの条件が真である場合、RED ALERT 文がレッド アラートを送信します。

```
ALARM gbl_cpu_total_util > 90
  OR
      gbl_cpu_sys_mode_util > 50 FOR 10 MINUTES
START
  RED ALERT "Either total CPU util or sys mode CPU high"
```

異なるインターバルで記録されるメトリックを、同じアラームで使用しないように注意してください。たとえば、次のように文内のグローバル メトリック (5 分のインターバルで記録) の値に基づいて、プロセス (1 分のインターバルで記録) についてループしないようにしてください。

```
IF global_metric THEN    PROCESS LOOP...
```

異なるインターバルは、予想どおりに同期しないため、結果は無効になります。

**注:** GlancePlus の場合、すべてのプロセスに対してアラームを送信するには、プロセス ループ内部のプロセス メトリックを使用してください。

# ALERT 文

ALERT 文により、Network Node Manager または Operations Manager のいずれかにメッセージを送信できます。ALERT 文は、ALARM 内での処理として最も頻繁に使用されます。また、IF 文内で ALERT 文を使用すると、継続時間の経過後ではなく、条件の検出後直ちにメッセージを送信できます。ALARM 文または IF 文以外で ALERT を使用すると、メッセージは各インターバルで送信されます。

## 構文

```
[RED, CRITICAL, ORANGE, MAJOR, YELLOW, MINOR, CYAN,WARNING, GREEN,
NORMAL, RESET] ALERT message
```

- RED は CRITICAL と、ORANGE は MAJOR、YELLOW は MINOR と、CYAN は WARNING と、GREEN は NORMAL と同義です。これらのキーワードは、アラーム シンボルをアラーム条件と関連付けられた色に変えます。
- RESET — ALARM 条件が終了すると、メッセージ付きの RESET ALERT を送信します。アラーム定義にリセットを定義していない場合は、ALARM 条件が終了すると、メッセージなしの RESET ALERT が送信されます。

- *message* — メッセージの作成に使用される文字列と数値の組み合わせです。数値のフォーマットは、`[|[-]width[|decimals]]` パラメータで設定できます。*width* はフォーマットするフィールドの幅を指定します。*decimals* は使用する小数の桁数を指定します。数値は右揃えです。- (マイナス記号) は左揃えを指定します。英数文字列は必ず左揃えです。

## 使い方

アラームジェネレータの構成方法により、ALERT は NNM への SNMP トラップの送信や、HPOM へのメッセージの送信をトリガできます。HPOM に送信されるアラートメッセージについては、WARNINGS がメッセージブラウザに青色で表示されます。

## 例

次に代表的な ALERT 文を示します。

```
RED ALERT "CPU utilization = ", gbl_cpu_total_util
```

Network Node Manager がある場合は、この文によって、Network Node Manager の Alarm Browser ウィンドウに重要度が危険域のアラームが作成されます。

# EXEC 文

EXEC 文により、ローカルシステム上で実行するシステム (UNIX または Windows) コマンドを指定できます。たとえば、EXEC 文を使用すると、一定の条件が満たされるたびに IT 管理者にメールを送信できます。

EXEC は、ALARM 文または IF 文内で使用されるため、コマンドは指定の条件が満たされたときにのみ実行されます。EXEC 文を ALARM 文または IF 文以外で使用すると、予測できないインターバルで動作が実行されます。

## 構文

`EXEC` *"system command"*

*system command* — ローカルシステム上で実行するコマンドです。

EXEC 文では二重引用符 (") を使用しないようにしてください。これは、`perfalarm` がアラームを HPOM に送信できなくなるためです。代わりに引用符 (') を使用してください。たとえば、次のようになります。

```
EXEC "echo 'performance problem detected' "
```

```
EXEC "mkdir c:\\directory\\filename"
```

EXEC 文の構文には、二重引用符で囲まれたファイルのパス名が必要です。ただし、パス名にスペースが含まれる場合、パス名を一重引用符で囲んでから、二重引用符でもう一度囲む必要があります。

例 :

```
EXEC "'C:\\Program Files\\Mail Program\\SendMail.exe'"
```

EXEC 文のシステムコマンドの引数に一重引用符が含まれている場合、EXEC 文でコマンドを実行するときに最初の一重引用符 (') のペアは二重引用符 (") に変換されるため、プログラムを一重引用符で囲む必要があります。

例 :

```
EXEC "'echo' 'test execution'"
```

前述の例では、echo はプログラムであり、引用符に囲まれた引数が含まれているため (この場合、test execution)、このプログラムは引用符で囲まれています。さらに、EXEC 文の構文により、コマンドの文字列全体を二重引用符で囲む必要があります。

EXEC 文では二重引用符 (") を使用しないようにしてください。perfalarm で HPOM へのアラームの送信が失敗します。代わりに一重引用符 (') を使用してください。

たとえば、次のようになります。

```
EXEC "'echo' 'dialog performance problem'"
```

前述の例では、echo はプログラムであり、引用符で囲まれた引数が含まれているため (この場合、dialog performance problem)、このプログラムは引用符で囲まれています。さらに、EXEC 文の構文により、コマンドの文字列全体を二重引用符で囲む必要があります。

## 使い方

EXEC は、アラームジェネレータの構成方法により、ローカルシステム上でローカルアクションをトリガできます。たとえば、ローカルアクションのオンとオフを切り替えられます。HPOM に情報を送信するようにアラームジェネレータを構成している場合、通常ローカルアクションは実行されません。

## 例

次の例では、`gbl_disk_util_peak` メトリックが 20 を超えたときに、EXEC 文が UNIX の `mailx` コマンドを実行します。

```
IF gbl_disk_util_peak > 20 THEN
          EXEC "echo 'high disk utilization detected'| mailx root"
```

次の例は、ネットワークパケット率が 15 分間の平均で 1 秒あたり 1000 を超えたときに、メールをシステム管理者に送信する EXEC 文を示しています。

```
ALARM gbl_net_packet_rate > 1000 for 15 minutes
    TYPE = "net busy"
    SEVERITY = 5
    START
      {
      RED ALERT "network is busy"
      EXEC "echo 'network busy condition detected'| mailx root"
      }
    END
    RESET ALERT "NETWORK OK"
```

**注 :** EXEC 文を頻繁に実行する場合、オーバヘッドが高いコマンドまたはスクリプトと共に EXEC

文を使用するときは、注意が必要です。

アラーム ジェネレータはコマンドを実行し、そのコマンドが完了するまで続行せずに待機します。完了するのに長時間かかるコマンドは指定しないように注意してください。

# PRINT 文

PRINT 文により、`utility` プログラムからその `analyze` 関数を使用してメッセージを印刷できます。アラーム ジェネレータは、PRINT 文を無視します。

## 構文

`PRINT` *message*

- *message*— メッセージを作成する文字列と数値の組み合わせです。数値のフォーマットは、`[|[-]`*width*`[|`*decimals*`]]` パラメータで設定できます。*width* はフォーマットするフィールドの幅を指定します。*decimals* は使用する小数の桁数を指定します。メッセージの英数字の構成要素は、引用符で囲む必要があります。数値は右揃えです。- (マイナス記号) は左揃えを指定します。英数文字列は必ず左揃えです。

## 例

```
PRINT "The total time the CPU was not idle is",
              gbl_cpu_total_time |6|2, "seconds"
```

この文を実行すると、次のようなメッセージが印刷されます。

```
The total time the CPU was not idle is 95.00 seconds
```

# IF 文

IF-THEN 論理を使用して条件を定義するには、IF 文を使用します。IF 文は、ALARM 文内で使用します。ただし、IF 文は単独で使用するか、IF-THEN 論理が必要な `alarmdef` ファイル内の任意の場所で使用します。

ALARM 文以外で IF 文を指定する場合、その実行頻度は制御できません。

## 構文

**IF** *condition* **THEN** *action* **[ELSE** *action***]**

`IF`*condition* — 条件は、2 つの項目間の比較として定義されます。

```
item1 {>, <, >=, <=, ==, !=}item2
   [AND, OR[item3 {>, <, >=, <=, ==, !=}item4]]
```

ここで、「`==`」は「等号」を、「`!=`」は「非等号」を意味します。

項目は、メトリック名、数値定数、引用符で囲まれた英数字の文字列、別名、変数のいずれかです。英数字の文字列を比較するときは、== または != のみを演算子として使用できます。

*action* ー 任意の動作、または変数を設定します(この場合、ALARM は無効です)。

## 使い方

IF 文は *condition* をテストします。*condition* が真である場合、THEN の後の *action* が実行されます。*condition* が偽である場合、*action* はオプションの ELSE 節に依存します。ELSE 節が指定されている場合、その後に続く *action* が実行されます。ELSE 節が指定されていない場合、IF 文は何も行いません。

## 例

この例では、CPU ボトルネックの兆候が計算され、ボトルネックが発生する可能性により、シアンまたは赤色の ALERT が定義されます。アラーム ジェネレータの構成方法に応じて、ALERT は NNM への SNMP トラップの送信や、CPU の利用率を伴った「`End of CPU Bottleneck Alert`」メッセージの Operations Manager への送信をトリガできます。

```
SYMPTOM CPU_Bottleneck > type=CPU
     RULE gbl_cpu_total_util  > 75  prob 25
     RULE gbl_cpu_total_util  > 85  prob 25
     RULE gbl_cpu_total_util  > 90  prob 25
     RULE gbl_cpu_total_util  > 4  prob 25

     ALARM CPU_Bottleneck > 50 for 5 minutes
       TYPE="CPU"
       START
         IF CPU_Bottleneck > 90 then
           RED ALERT "CPU Bottleneck probability= ",
                     CPU_Bottleneck, "%"
         ELSE
           CYAN ALERT "CPU Bottleneck probability= ",
                     CPU_Bottleneck, "%"
       REPEAT every 10 minutes
         IF CPU_Bottleneck > 90 then
           RED ALERT "CPU Bottleneck probability= ",
                     CPU_Bottleneck, "%"
         ELSE
           CYAN ALERT "CPU Bottleneck probability= ",
                     CPU_Bottleneck, "%"
       END
          RESET ALERT "End of CPU Bottleneck Alert"
```

異なるインターバルで記録されるメトリックを、同じ文で使用しないように注意してください。たとえば、次のように文内のグローバル メトリック (5 分間のインターバルで記録) の値に基づいて、プロセス (1 分間のインターバルで記録) についてループしないようにしてください。

```
IF global_metric THEN PROCESS LOOP ...
```

異なるインターバルは、予想どおりに同期しないため、結果は無効になります。

# LOOP 文

`LOOP` 文は複数インスタンス データ型を調べ、各インスタンスに定義された *action* を実行します。

## 構文

**{APPLICATION, PROCESS, LVOLUME, DISK, CPU, FILESYSTEM, TRANSACTION, NETIF, LOGICAL} LOOP**

*action*

- APPLICATION, PROCESS, LVOLUME, DISK, CPU, FILESYSTEM, TRANSACTION, NETIF, LOGICAL — 複数インスタンスデータを含む Performance Collection Component のデータ型です。
- *action* — PRINT、EXEC、ALERT、集合変数です。

## 使い方

LOOP 文がデータ型の各インスタンスを繰り返すと、メトリック値が変わります。たとえば、utility プログラムの analyze コマンドを使用している場合、次の LOOP 文は各アプリケーションの名前を stdout に出力します。

```
APPLICATION LOOP

PRINT app_name
```

LOOP は、最高 5 レベルまで別の LOOP 文内にネストできます。

LOOP を実行するには、LOOP 文が LOOP 文で定義されている型と同じデータ型のメトリックを 1 つ以上参照する必要があります。

## 例

アクティブなすべてのアプリケーションを繰り返すときには、LOOP 文を使用します。

次の例では、各インターバルで CPU 利用率が最大のアプリケーションを検索する方法を示しています。

```
highest_cpu = 0
 APPLICATION loop
   IF app_cpu_total_util > highest_cpu THEN
    {
      highest_cpu = app_cpu_total_util
      big_app = app_name
     }

 ALERT "Application ", app_name, " has the highest cpu util at
",highest_cpu_util|5|2, "%"

 ALARM highest_cpu > 50
   START
     RED ALERT  big_app, " is the highest CPU user at ", highest_cpu,
"%"
   REPEAT EVERY 15 minutes
     CYAN ALERT big_app, " is the highest CPU user at ", highest_cpu,
```

```
"%"
   END
 RESET ALERT "No applications using excessive cpu"
```

# INCLUDE 文

INCLUDE 文は、デフォルトの alarmdef ファイルと共に別のアラーム定義ファイルを含めるときに使用します。

## 構文

INCLUDE *"filename"*

ここで、*filename* は別のアラーム定義ファイルの名前です。ファイル名は、必ず完全修飾名にします。

## 使い方

INCLUDE 文を使用すると、論理的に異質なアラーム定義一式を別々のファイルに分離できます。

## 例

たとえば、独立したファイルにトランザクション メトリックに関するいくつかのアラーム定義があり、そのファイルに次のような名前が付いていると仮定します。

trans_alarmdef1

alarmdef1 ファイルのアラーム定義に次の行を追加すると、alarmdef1 ファイルを使用できます。

INCLUDE "/var/opt/perf/trans_alarmdef1"

# USE 文

デフォルトの SCOPE データソース以外のデータソースを参照する場合は、alarmdef ファイルのメトリック名の使い方を単純化するために、USE 文を追加できます。これにより、データソース名を使用せずにメトリック名を指定できます。

データソース名は datasources ファイルで定義する必要があります。無効なデータソース名が見つかると、alarmdef ファイルの構文の確認が失敗します。

> **注 :** alarmdef ファイルに USE 文があっても、USE 文に続くすべてのメトリック名が指定のデータソースから得られるわけではありません。

## 構文

USE *"datasourcename"*

## 使い方

アラーム ジェネレータは、alarmdef ファイルの構文が有効であることを確認するときに、ファイル内で参照する順序に従ってすべてのデータソースの検索リストを作成します。メトリックの完全修飾名または USE 文が見つかると、Perfalarm はエントリを順番にこのデータソース検索リストに追加します。その後、このリストは完全に修飾されていないメトリック名を適切なデータソース名に適合させるために使用されます。USE 文は、データソースを perfalarm の検索リストに追加する便利な方法です。これにより、alarmdef ファイルのメトリック名は短縮できます。メトリック名の構文については、この章で前述している「メトリック名」を参照してください。

メトリック名をデータソースに適合させるための Perfalarm のデフォルトの動作では、最初に SCOPE データソースでメトリック名を検索します。perfalarm が alarmdef ファイルの最初のメトリック名を見つけると、暗黙的に指定されている USE "SCOPE" 文が実行されます。この機能により、alarmdef ファイルの SCOPE のメトリックが完全に修飾されていなくても参照できるように、SCOPE データソースに対してデフォルトの検索パスが設定されます。次のページに、この例を示します。

```
ALARM gbl_cpu_total_util > 80 FOR 10 MINUTES
     START RED ALERT "CPU utilization too high"

USE "ORACLE7"

ALARM ActiveTransactions >= 95 FOR 5 MINUTES
      START RED ALERT "Nearing limit of transactions for ORACLE7"
```

perfalarm が上記の文を含む alarmdef ファイルの構文を確認すると、メトリック「gbl_cpu_total_util」が見つかり、次にそのデータソースを検索します。Perfalarm のデータソース検索リストにまだデータソースがないため、暗黙的に指定されている USE "SCOPE" 文を実行し、SCOPE データソースでメトリック名を検索します。適合するメトリック名を見つけた後、引き続き perfalarm は alarmdef ファイルの残りのデータを確認します。

USE "ORACLE7" 文を見つけると、perfalarm はデータソース検索リストに ORACLE7 データソースを追加します。メトリック名 "ActiveTransactions" が見つかると、perfalarm は SCOPE データソースからデータソースのリストを順に検索します。SCOPE にそのメトリック名が含まれない場合、次に ORACLE7 データソースが検索され、適合するメトリック名を見つけます。

perfalarm により、適合するメトリック名がどのデータソースでも見つからなかった場合は、エラーメッセージが出力され、perfalarm が終了します。

デフォルトの検索動作を変更するには、メトリック名を参照する前に、alarmdef ファイルの先頭に USE 文を追加します。これにより、USE 文で指定されているデータソースが、SCOPE データソースの前に、データソース検索リストに追加されます。USE 文内のデータソースは、適合するメトリック名を見つけるために、SCOPE データソースの前に検索されます。次に、この例を示します。

USE 文によってデータソースが参照されると、データソースの順序を変更したり、検索リストからデータソースを削除することはできません。

```
USE "ORACLE7"

ALARM gbl_cpu_total_util > 80 FOR 10 MINUTES
      START RED ALERT "CPU utilization too high"

ALARM ActiveTransactions >= 95 FOR 5 MINUTES
      START RED ALERT "Nearing limit of
      transactions for ORACLE7"
```

上記の例では、alarmdef ファイル内の文の順序が変更されています。メトリック名が参照される前に USE "ORACLE7" 文が定義されるため、データソース検索リストの最初のデータソースとして ORACLE7 データソースが追加されます。perfalarm が最初のメトリック名 "gbl_cpu_total_util" を見つけると、暗黙的に指定されている USE "SCOPE" 文が実行されます。"gbl_cpu_total_util" メトリック名は完全修飾名ではないため、perfalarm は ORACLE7 からデータソースのリストを順に検索します。SCOPE にそのメトリック名が含まれない場合、次に ORACLE7 データソースが検索され、適合するメトリック名を見つけます。

引き続き perfalarm は alarmdef ファイルの残りのデータを確認します。"ActiveTransactions" メトリックが見つかると、perfalarm は ORACLE7 からデータソースのリストを検索します。適合するメトリック名を見つけた後、perfalarm は引き続き alarmdef ファイルの残りのデータを検索します。perfalarm により、適合するメトリック名 (完全に修飾されていないメトリック名 ) がすべてのデータソースで見つからなかった場合は、エラーメッセージが出力され、perfalarm が終了します。

複数のデータソースに同じメトリック名が含まれる場合、注意して USE 文を使用してください。perfalarm はデータソースのリストを順番に検索します。同じメトリック名を使用する異なるデータソースのアラーム条件を定義している場合、正しいデータソースからメトリックの値が検索されるように、データソース名でメトリック名を修飾する必要があります。次の例を参照してください。アラーム文内の各メトリック名には、データソースが含まれています。

```
ALARM ORACLE7:ActiveTransactions >= 95 FOR 5 MINUTES
   START RED ALERT "Nearing limit of transactions for ORACLE7"

ALARM FINANCE:ActiveTransactions >= 95 FOR 5 MINUTES
   START RED ALERT "Nearing limit of transactions for FINANCE"
```

# VAR 文

VAR 文により、変数を定義し、その変数に値を割り当てることができます。

## 構文

[VAR] *name* = *value*

*name* — 変数名は必ず英字から始まり、英数字とアンダースコア文字を使用できます。変数名に大文字と小文字の区別はありません。

*value* — 値が英数字の文字列の場合は、引用符で囲む必要があります。

## 使い方

VAR は、値をユーザー変数に割り当てます。変数が事前に存在しない場合は、作成します。

定義した変数は、alarmdef ファイルの任意の場所で使用できます。

## 例

変数は、何かを割り当てることによって定義します。次の例では、*highest_CPU_value* に 0 の値を割り当てることにより、数値変数を定義しています。

```
highest_CPU_value = 0
```

次の例では、*my_name* に空の文字列値を割り当てることにより、英数字変数を定義しています。

```
my_name = ""
```

# ALIAS 文

メトリック名の部分 (クラス、インスタンス、メトリックのいずれか) に大文字と小文字を区別する名前がある場合や、特殊文字を含む名前がある場合、ALIAS 文により別名に置換できます。ALIAS 文はこのような場合にのみ使用します。

## 構文

ALIAS *name* =*"replaced-name"*

- *name* ー 名前は必ず英字から始まり、英数字とアンダースコア文字を使用できます。
- *replaced-name* ー アラームジェネレータが一意に認識できるように、ALIAS 文で置換する必要がある名前です。

## 使い方

alarmdef ファイルの処理方法によって、メトリック名の一部 (クラス、インスタンス、メトリック名のいずれか) が、大文字と小文字を認識することで一意に識別される場合は、別名を作成する必要があります。また、特殊文字を含む名前についても、別名を作成する必要があります。たとえば、アプリケーション「BIG」と「big」がある場合、「big」に別名を定義して、別々のアプリケーションとして表示されるようにする必要があります。別名は、alarmdef ファイル内で、置換する名前の最初のインスタンスより前に定義する必要があります。

## 例

構文では特殊文字や大文字と小文字を使用できないため、アプリケーション名「AppA」と「appa」を使用するとエラーが発生します。これは、処理がこれらの名前を区別できないためです。「AppA」の別名を定義して、一意に認識可能な名前を付けます。たとえば、次のようになります。

```
ALIAS appa_uc = "AppA"
ALERT "CPU alert for AppA.util is",appa_uc:app_cpu_total_util
```

クラス識別名のある別名を使用する場合、インスタンス名とクラス名の両方が含まれる別名である必要があります。次の例は、インスタンス名が 'other'、クラス名が 'APPLICATION.' の別名を示しています。

```
ALIAS my_app="other(APPLICATION)"
```

```
ALERT my_app:app_cpu_total_util > 50 for 5 minutes
```

# SYMPTOM 文

SYMPTOM により、条件一式に基づいて単一の変数値を設定する方法が提供されます。すべての条件が真である場合、必ずその確率値が SYMPTOM の変数値に追加されます。

## 構文

SYMPTOM *variable*

RULE*condition* PROB *probability*

[RULE *condition* PROB *probability*]

.

.

.

- SYMPTOM キーワードおよび RULE キーワードは、SYMPTOM 文でのみ使用し、別の構文では使用できません。SYMPTOM 文は、トップレベルの文でなければならず、他の文内にネストできません。SYMPTOM の後の他の文は、SYMPTOM に対応するすべての RULE 文が終了するまで、処理されません。
- *variable* は、この兆候の名前になる変数名です。SYMPTOM 文で定義された変数名は別の構文で使用できますが、それらの構文で変数値を変更しないようにしてください。
- RULE は SYMPTOM 文のオプションであり、独立して使用できません。RULE オプションは SYMPTOM 文内で必要な数だけ使用できます。SYMPTOM 変数は、各インターバルで規則に従って評価されます。
- *condition* は、2 つの項目間の比較として定義されます。
  item1 {>, <, >=, <=, ==, !=}item2

  [item3 {>, <, >=, <=, ==, !=}item4]

  ここで、「==」は「等号」を、「!=」は「非等号」を意味します。

  項目は、メトリック名、数値定数、引用符で囲まれた英数字の文字列、別名、変数のいずれかです。英数字の項目を比較するときは、== または != のみを演算子として使用できます。
- *probability* は数値定数です。真である各 SYMPTOM RULE の確率を加算して、SYMPTOM 値を作成します。

## 使い方

測定値と値との間の条件が真であるすべての確率の合計が、その兆候が起きている確率になります。

## 例

```
SYMPTOM CPU_Bottleneck
RULE gbl_cpu_total_util > 75  PROB 25
RULE gbl_cpu_total_util > 85  PROB 25
RULE gbl_cpu_total_util > 90  PROB 25
RULE gbl_run_queue      >  3  PROB 50
IF CPU bottleneck > 50 THEN
CYAN ALERT "The CPU symptom is: ", CPU_bottleneck
```

# アラーム定義の例

次の例は、一般的なアラーム定義の使用例です。

## CPU の問題の例

この例では、CPU 使用率が 5 分間にわたって 90 % を超え、CPU 実行待ち行列が 5 分間 3 を超えた場合に、アラーム ジェネレータの構成方法に応じて、Network Node Manager に SNMP トラップを送信したり、Operations Manager にメッセージを送信します。

```
ALARM gbl_cpu_total_util > 90 AND
  gbl_run_queue > 3 FOR 5 MINUTES
START
  CYAN ALERT "CPU too high at", gbl_cpu_total_util, "%"
REPEAT EVERY 20 MINUTES
{
  RED ALERT "CPU still to high at ", gbl_cpu_total_util, "%"
  EXEC "/usr/bin/pager -n 555-3456"
}
END
  RESET ALERT "CPU at ", gbl_cpu_total_util, "% - RELAX"
```

両方の条件が 20 分後も真のままであれば、重要度が危険域のアラームが NNM に作成されます。次にプログラムが実行され、システム管理者をページングします。

アラーム条件のいずれか 1 つが真でなくなった場合、アラーム シンボルが削除され、メッセージが送信されて、グローバル CPU 使用率、アラートの終了時間、`RELAX` への注意が表示されます。

## スワップ利用率の例

この例では、スワップ スペースの利用率が 5 分間 95% を超えた場合、アラーム ジェネレータの構成方法により、ALERT は NNM への SNMP トラップの送信や、HPOM へのメッセージの送信をトリガできます。

```
ALARM gbl_swap_space_util > 95 FOR 5 MINUTES
  START
    RED ALERT "GLOBAL SWAP space is nearly full"
  END
    RESET ALERT "End of GLOBAL SWAP full condition"
```

## 時間に起因するアラームの例

アラームの状態がアクティブであるようにインターバルを指定できます。たとえば、正規のインターバルで実行するように指定されたシステムの保守作業を行う場合、通常の操作時間のアラーム状態と、それとは異なるセットでシステム保守時間のアラーム状態を指定できます。

この例では、8:00 AM から 5:00 PM の間にのみアラームはトリガされます。

```
start_shift = "08:00"
end_shift = "17:00"
```

```
ALARM gbl_cpu_total_util > 80
    TIME > start_shift
    TIME < end_shift for 10 minutes
  TYPE = "cpu"
  START
    CYAN ALERT "cpu too high at ", gbl_cpu_total_util, "%"
  REPEAT EVERY 10 minutes
    RED ALERT"cpu still too high at ", gbl_cpu_total_util, "%"
  END
    IF time == end_shift then
    {
    IF gbl_cpu_total_util > 80 then
       RESET ALERT "cpu still too high, but at the end of shift"
    ELSE
       RESET ALERT  "cpu back to normal"
    ELSE
```

## ディスク インスタンスのアラームの例

アラームは、特定のディスク インスタンス名と対応するメトリック名を指定すると、アラームを特定のディスクに生成できます。

アラーム構文の次の例では、特定のディスク インスタンスのアラームが生成されます。ディスク インスタンスで特殊文字が使用されている場合、別名を使用する必要があります。

```
ALIAS diskname=""
ALARM diskname:bydsk_phys_read > 1000 for 5 minutes
TYPE="Disk"
  START
     RED ALERT "Disk  "
  REPEAT EVERY 10 MINUTES
     CYAN ALERT "Disk  cyan alert"
  END
     RESET ALERT "Disk  reset alert"
```

# アラーム定義のカスタマイズ

アラーム定義ファイル `alarmdef` でアラームを発生させる条件を指定します。Performance Collection Component の初期インストール時、`alarmdef` ファイルにはデフォルトのアラーム定義セットが書き込まれます。これらのデフォルトのアラーム定義は、そのまま使用できますが、必要に応じてカスタマイズもできます。

`alarmdef` ファイルは次の手順でカスタマイズします。

1. 必要に応じてアラームの定義を改訂します。この章のアラーム定義構文の例を参照してください。
2. ファイルを保存します。
3. Performance Collection Componentの `utility` プログラムを使用して、アラームの定義の妥当性を検査します。

a. `utility` と入力します。

b. プロンプトで次のように入力します。

```
checkdef
```

これにより、アラームの構文が確認され、ファイルに問題がある場合は、エラーまたは警告が表示されます。

4. 新しいアラームの定義を有効にするために、次のように入力します。

```
ovpa restart alarm
```

または

```
mwa restart alarm
```

これにより、アラーム ジェネレータが終了して再起動し、カスタマイズした `alarmdef` ファイルを読み取ります。

Performance Collection Component の各システムに固有のアラーム定義セットを使用するか、グループ全体で同じアラーム定義セットを使用することによりシステムのグループの監視作業を標準化できます。

`alarmdef` ファイルが大きくなると、Performance Collection Component アラーム ジェネレータおよび `utility` プログラムが正しく動作しないことがあります。この問題は、システム リソースの空き容量に起因する場合があります。

パフォーマンス アラームについて習得するには、新しいアラーム定義の追加や、デフォルトのアラーム定義の変更を試すことが最良の方法といえます。

# RTMA コンポーネントのアドバイザ

アドバイザ機能を使用するには、HP Ops OS Inst to Realtime Inst LTU または Glance Pak Software LTU を有効にする必要があります。

アドバイザ機能を使用すると、RTMA コンポーネントによって収集された一部のメトリックの値が設定したしきい値を超過した (または下回った) 場合に、アラームを生成、表示できます。アドバイザ機能は、**adviser** スクリプトと **padv** ユーティリティで構築されます。adviser スクリプトは、監視対象システムのパフォーマンスの低下の兆候が見られた際に、アラームを生成するといったルールを作るのに便利です。`padv` ユーティリティは、対象のシステムで adviser スクリプトを実行するのに使用します。

> **注 :** このトピックでは、RTMA コンポーネントで使用可能なアドバイザ機能を詳述します。GlancePlus ソフトウェアには、アドバイザユーティリティで使用する追加の機能が備わっています。GlancePlus ソフトウェアでのアドバイザ機能の使用方法については、*GlancePlus* のオンライン ヘルプを参照してください。

## アラームと兆候

アラームを使用すると、メトリック条件を強調表示できます。adviser スクリプトでは、RTMA コンポーネントによって監視するメトリックのしきい値を定義できます。メトリックの値が設定したしきい値を超えると、RTMA コンポーネントはアラーム メッセージの形式でアラームを生成します。このメッセージは `stdout` の形で **padv** ユーティリティに送信されます。

指定した条件を満たした場合に常にアラームを生成するよう設定できます。アラームは指定した期間 (1 つのまたは複数のインターバル) に基づいて生成されます。

兆候は、システムのパフォーマンスに影響を与える条件を組み合わせたものです。

さまざまなメトリックを対応するしきい値で観察し、ボトルネックの原因となっている確率に値を追加することにより、アドバイザはボトルネックが発生する複合的な確率を表す値を計算します。

## adviser スクリプトでの作業

`padv` コマンドを実行すると、HP Operations エージェントはコマンドで指定したスクリプトの走査を行い、必要なアクションを実行します。`padv` コマンドでスクリプト ファイルを指定しない場合、アドバイザ ユーティリティは以下のデフォルトのスクリプト ファイルから必要な情報を取得します。

- *Windows の場合 :%ovdatadir%*`\perf\perfd`
- *UNIX/Linux の場合 :* `/var/opt/perf/perfd`

オペレーティング システム固有の診断およびアクションを含むスクリプトを実行する場合は、以下のデフォルトのスクリプトを使用します。

- *Windows の場合 :%ovdatadir%*`\perf\perfd\os\`*<OS の種類>*`\adv`
- *UNIX/Linux の場合 :* `/var/opt/perf/perfd/os/`*<OS の種類>*`/adv`

この例では、*<os_type>* はスクリプトを実行するノードのオペレーティング システムのタイプになります。

adviser スクリプトを実行した結果、以下の作業を実行できます。

- 生成されたアラームに基づいてシステムのステータスをテキスト ファイルに出力
- padv コマンドを実行したコマンド コンソールにシステムのリアルタイムのステータスを表示

# アドバイザの使用

アドバイザコンポーネントを使用してシステムのリアルタイムのヘルスを監視するには、以下の手順を実行します。

1. ご使用の環境に合わせて adviser スクリプトを構成します。以下のディレクトリにはサンプルのスクリプトが用意されています。
   - *Windows の場合:%ovinstalldir%*`\examples\adviser`
   - *UNIX または Linux 環境:* `/opt/perf/examples/adviser`
2. スクリプトを実行するノードを特定します。
3. 特定したシステムで `perfd` プロセスが実行されていることを確認します。
4. 次のコマンドを実行します。

   `padv -s` *<script_name>*`-n`*<system_name>*

   指定したシステムで adviser スクリプトが開始され、スクリプト ファイルの構成に応じて結果が作成されます。

   **ヒント:** リモート システムでスクリプトを使用するときは、リモート システムで `perfd` プロセスが実行されている必要があります。他のシステムのローカルで adviser スクリプトを実行しないようにできます。詳細情報は、Restrict Access on page 57を参照してください。

## 複数のシステム上での adviser スクリプトの実行

`mpadv` コマンドを使用すると、複数のシステム上で adviser スクリプトを実行できます。`mpadv` コマンドを使用するには、以下の手順を実行します。

1. スクリプトを実行するノードを特定します。
2. 特定したすべてのシステムの名前をリストしたテキスト ファイルを作成します。
3. ローカル システムにテキスト ファイルを保存します。
4. ご使用の環境に合わせて adviser スクリプトを構成します。以下のディレクトリにはサンプルのスクリプトが用意されています。
   - *Windows オペレーティング システム: %ovinstalldir%*`\examples\adviser`
   - *UNIX または Linux 環境:* `/opt/perf/examples/adviser`
5. 特定したシステムで `perfd` プロセスが実行されていることを確認します。
6. 次のコマンドを実行します。

   `mpadv -l` *<system_list_text_file>* `-s` *<script_name>*

   *<system_list_text_file>* ファイルで指定したシステムで adviser スクリプトが開始され、スクリプト ファイルの構成に応じて結果が作成されます。

# アドバイザ構文

アドバイザ構文は簡単なスクリプト言語で、アラームの設定および兆候の条件の定義が可能です。

デフォルトの構文ファイル、`adviser.syntax` は以下のディレクトリにあります。

- *Windows オペレーティング システム:%ovdatadir%*`\perf`
- *UNIX および Linux オペレーティング システム:*`/var/opt/perf`

構文ファイルを編集することで、独自のアラームと兆候を定義できます。

## 構文表記

- 中かっこ ({}) は、項目の 1 つを選択することを示します。
- 大かっこ ([ ]) は、オプションの項目を示します。
- 大かっこまたは中かっこ内でカンマで区切られている項目はオプションです。1 つだけ選択してください。
- 斜体は、置換する変数名を示します。
- アドバイザ構文のキーワードは大文字にする必要があります。

## コメント

構文:

# *[任意のテキストまたは文字列]*

または

// *[任意のテキストまたは文字列]*

コメントの前には、二重スラッシュ (//) または # 記号を付けます。どちらの場合も、コメントはその行の終わりで終了します。

## 条件

条件は、2 つのメトリック名、ユーザー変数、数値定数間の比較として定義されます。

```
item1 {>, <, >=, <=, ==, !=} item2 [OR item3 \
                    {>, <, >=, <=, ==, !=} item4]
```

または:

```
item1 {>, <, >=, <=, ==, !=} item2 [AND item3 \
                    {>, <, >=, <=, ==, !=} item4]
```

(「==」は「等号」を、「!=」は「非等号」を意味します。)

条件は、ALARM 文、IF 文で使用されます。2 つの数値メトリック、変数または定数を比較できるほか、2 つの文字列メトリック名、ユーザー変数、文字列定数にも使用できます。文字列の条件の場合は、== または != のみを演算子として使用できます。

二次条件間で演算子のORとANDを指定して、複合条件を使用できます。

例:

```
gbl_swap_space_reserved_util > 95

proc_proc_name == "test" OR proc_user_name == "tester"

proc_proc_name != "test" AND

                  proc_cpu_sys_mode_util > highest_proc_so_far
```

## 定数

定数は、英数字または数値です。英数字の定数は、二重引用符で囲む必要があります。数値定数には、整数または実数の2種類があります。整数定数には数字とオプションの符号標識を使用できます。実数定数には小数点も含めることができます

例:

| | |
|---|---|
| 345 | 整数 |
| 345.2 | 実数 |
| “Time is” | 英数字リテラル |

## 式

式を使用すれば数値を評価できます。式は、条件またはアクション内で使用できます。

式には以下のものを含めることができます。

- 数値定数
- 数値メトリック名
- 数値変数
- 上記の算術比較
- かっこを使用して上記をグループ化したものの組み合わせ

例:

```
Iteration + 1

3.1416

gbl_cpu_total_util - gbl_cpu_user_mode_util

( 100 - gbl_cpu_total_util ) / 100.0
```

## アドバイザ構文内のメトリック名

アドバイザ構文内のメトリックはその位置にかかわらず直接参照できます。アドバイザ構文には以下のタイプのメトリックを使用できます。

- グローバル メトリック (接 頭 辞 gbl_ または tbl_)
- アプリケーション メトリック (接 頭 辞 app_)
- プロセス メトリック (接 頭 辞 proc_)
- ディスク メトリック (接 頭 辞 bydsk_)
- CPU ごとのメトリック (接 頭 辞 bycpu_)
- ファイル システム メトリック (接 頭 辞 fs_)
- 論 理 ボリューム メトリック (接 頭 辞 lv_)
- ネット ワーク インターフェイス メトリック (接 頭 辞 bynetif_)
- スワップ メトリック (接 頭 辞 byswp_)
- ARM メトリック (接 頭 辞 tt_ または ttbin_)
- PRM メトリック (接 頭 辞 prm_)
- ローカリティ ドメイン メトリック (接 頭 辞 ldom_)

LOOP 文 のコンテキストには、プロセス、論 理 ボリューム、ディスク、ファイル システム、LAN、スワップの各 メトリックのみが使 用 できます。

メトリックには英 数 字 (gbl_machine または app_name など) または数 値 データを含 めることができますので、異 なる種 類 の測 定 を反 映 させることができます。たとえば、以 下 のようにメトリック名 の末 尾 は測 定 対 象 を示 します。

- `a _util` メトリック。利 用 率 をパーセンテージで測 定
- `a _rate` メトリック。1 秒 あたりの単 位 を計 測
- `a _queue` メトリック。リソースを待 っているプロセスまたはスレッド の数 を計 測

特 定 のメトリックの計 測 単 位 がわからない場 合 は、メトリック定 義 ド キュメントを参 照 してください。

LOOP 文 を使 用 しているとき以 外 は、アプリケーション メトリックに特 定 のアプリケーションを関 連 付 ける必 要 があります。この方 法 は、アプリケーション名 を指 定 し、その後 にコロンを入 力 し、続 いてメトリック名 を入 力 します。たとえば、other_apps:app_cpu_total_util は、アプリケーション other_apps の合 計 CPU 使 用 率 を指 定 します。構 文 内 でのアプリケーション メトリックの使 用 方 法 の詳 細 は、ALIAS 文 の説 明 を参 照 してください。

parm ファイルによって定 義 されることから、アプリケーション名 には特 殊 文 字 およびスペースを使 用 できます。構 文 でこれらの名 前 を使 用 するには (アプリケーション名 を変 数 名 の形 と一 致 させる必 要 があります)、大 文 字 と小 文 字 を区 別 せずに名 前 を作 成 し、スペースをアンダースコアに変 換 します。つまり、「Other Apps」と定 義 されたアプリケーション名 は、構 文 内 では `other_apps` とも参 照 されます。特 殊 文 字 を使 用 して定 義 されたアプリケーション名 の場 合 、ALIAS 文 を使 用 して別 名 を指 定 する必 要 があります。

明 示 的 に指 定 されている場 合 は、アプリケーション メトリックは構 文 内 のあらゆる場 所 で参 照 可 能 となります。指 定 されていないアプリケーション メトリックは、LOOP 文 のコンテキスト内 でのみ参 照 されます。LOOP 文 は、アプリケーション メトリックまたはプロセス メトリックを暗 黙 的 に指 定 する反 復 文 です。

LOOP 文 のコンテキスト内 でのみ、プロセス メトリックを参 照 できます。プロセスを明 示 的 に参 照 する方 法 はありません。

# 出力リスト

出力リストは、正しくフォーマットされた式、メトリック名、ユーザー変数、定数の組み合わせです。正しいフォーマットについては、次の例を参照してください。

- 式の例:

```
expression [|width[|decimals]]
```

メトリック名またはユーザー変数の例:

```
metric names [|width[|decimals]]
```

または

*user variables [|width[|decimals]]*

メトリック名またはユーザー変数は英数字で指定します。

- 定数の例:

定数にはフォーマット化は必要ありません。

フォーマット化の例:

```
gbl_cpu_total_util|6|2    formats as  '100.00'
(100.32 + 20)|6    formats as     '   120'
gbl_machine|5      formats as         '7013/'
"User Label"       formats as       "User Label"
```

# 変数

変数は必ず英字から始まり、英数字とアンダースコア文字を含められます。変数では、大文字と小文字を区別しません。

変数は、何かを割り当てることにより定義します。次の例では、highest_CPU_value に 0 の値を割り当てることにより、数値変数を定義しています。

```
highest_CPU_value = 0
```

次の例では、my_name に null の文字列値を割り当てることにより、英数字変数を定義しています。

```
my_name = ""
```

# アドバイザ構文

## ALARM 文

ALARM 文を使用すると、定義した一部のイベントがシステムで発生した際に通知を行います。ALARM 文を使用すると、adviser スクリプトで padv コマンドの発信元コンソールにメッセージを送信して通知するよう設定できます。

構文:

```
ALARM condition [FOR duration {SECONDS, MINUTES, INTERVALS}]
 [condition [FOR duration {SECONDS, MINUTES, INTERVALS}] ] ...

[START statement]

[REPEAT [EVERY duration [SECONDS, MINUTES, INTERVAL, INTERVALS]]
 statement]

[END statement]
```

ALARM 文は、トップレベルの文である必要があります。ALARM 文は別の文にネストできません。

ただし、いくつかの ALARM 条件を単一の ALARM 文に含めることはできます。この場合、トリガとなるためには、すべての条件が真となる必要があります。また、アラーム サイクル中の適切な時間に実行される COMPOUND 文も使用できます。

START、REPEAT、END は、ALARM 文のキーワードです。これらの各キーワードが文を指定します。ALARM 文には START、REPEAT、END が必要で、正しい順番にリストします。

アラーム サイクルは、すべてのアラーム条件、少なくとも指定の継続時間真であった最初のインターバルに開始します。その時間に、adviser スクリプトは、START 文を実行し、その後の各インターバルで REPEAT 条件を確認します。十分な時間が確認されると、REPEAT 節の文が実行されます。この処理は 1 つ以上のアラーム条件が偽になるまで続きます。この処理がアラーム サイクルを完了し、END 文が実行されます。

REPEAT 文で EVERY 仕様が省略されている場合、adviser スクリプトは各インターバルで REPEAT 文を実行します。

## ALERT 文

ALERT 文を使用すると、`padv` コマンド コンソールにメッセージを配置できます。ALARM が障害を検知した場合は、ALERT 文を実行して、指定した重要度のメッセージを `padv` コマンド コンソールに送信するよう設定できます。

ALERT 文は ALARM 文と組み合わせて使用できます。

構文:

```
[(RED or CRITICAL), (YELLOW or WARNING), RESET] ALERT printlist
```

RED and YELLOW, are synonymous with CRITICAL and WARNING.

## ALIAS 文

ALIAS 文を使用すると、特殊文字または空白を含むアプリケーション名に変数を割り当てることができます。

構文:

```
ALIAS variable = "alias name"
```

ALIAS の例

構文では特殊文字や空白を使用できないため、下記の PRINT 文でアプリケーション名「other user root」を使用するとエラーが発生します。ALIAS を使用すれば、「other user root」や特殊文字や空白を持つ他の文字列も構文で使用できます。

```
ALIAS otherapp = "other user root"


PRINT "CPU for other root login processes is: ",

 otherapp:app_cpu_total_util
```

## ASSIGNMENT 文

ASSIGNMENT 文を使用すると、ユーザー変数に数値、英数字、式を割り当てられます。

構文:

```
[VAR] variable = expression

[VAR] variable = alphaitem

[VAR] variable = alphaitem
```

## COMPOUND 文

COMPOUND 文は、IF 文や LOOP 文、ALARM 文の START、REPEAT、END の各節と共に使用します。COMPOUND 文を使用すると、リスト化した文を実行できます。

構文

```
{

statement

statement

}
```

COMPOUND 文を作成するには、中かっこ ({ }) 内に文をグループ化します。こうすることで、COMPOUND 文は構文内で単一の文として処理されます。

COMPOUND 文には、ALARM 文および SYMPTOM 文は含められません。(COMPOUND は文のタイプであり、キーワードではありません。)

## EXEC 文

EXEC 文を使用すると、アドバイザ構文から UNIX コマンドを実行できます。たとえば、ある条件を満たすたびに MIS スタッフに対してメールメッセージを送信する場合などに、EXEC コマンドを使用します。

構文

```
EXEC printlist
```

結果の出力リストは、実行用としてオペレーティングシステムに送信されます。

指定した EXEC コマンドは更新間隔ごとに 1 回実行されますので、オーバーヘッドの高くなるオペレーティングシステムのコマンドまたはスクリプトと EXEC 文を併用する際には注意が必要です。

## IF 文

IF 文を使用すると、adviser スクリプト構文に定義した条件をテストできます。

構文:

```
IF condition THEN statement [ELSE statement]
```

IF 文 は condition をテストします。condition が真である場合、THEN の後の文が実行されます。condition が偽である場合、動作はオプションの ELSE 節に依存します。

ELSE 節が指定されている場合、その後に続く文が実行されます。ELSE 節が指定されていない場合、IF 文は何も行いません。文を COMPOUND 文にすれば、adviser スクリプトで複数の文を実行するように設定できます。

## LOOP 文

LOOP 文を使用すると、システムに関する情報を確認できます。たとえば、CPU 使用率の最も高いプロセス、最も頻繁に使用されているスワップ領域などを確認できます。この情報を確認するには、LOOP 文のほか、情報収集対象のシステム条件にメトリック名を使用するそれぞれの文を使用します。

構文:

```
{APPLICATION, APP, CPU, DISK, DISK_DETAIL, FILESYSTEM, FS, FS_DETAIL,
LAN,

LOGICALVOLUME, LV, LV_DETAIL, NETIF, NFS, NFS_BYSYS_OPS, NFS_OP, PRM,

PRM_BYVG, PROCESS, PROC, PROC_FILE, PROC_REGION, PROC_SYSCALL, SWAP,

SYSTEMCALL, SC, THREAD, TRANSACTION, TT, TTBIN, TT_CLIENT, TT_
INSTANCE,

TT_UDM, TT_RESOURCE, TT_INSTANCE_CLIENT, TT_INSTANCE_UDM, TT_CLIENT_
UDM,

LDOM, PROC_LDOM}
```

LOOP 文

LOOP は、最高 5 レベルまで他の構文内にネストできます。ループの各反復で実行される文のブロックを含む COMPOUND 文として設定することもできます。BREAK 文を指定することで、LOOP 文からエスケープできます。

特定のデータを収集する構文内に LOOP 文を設定しても、システムに対応するメトリックデータがない場合は、adviser スクリプトは LOOP をスキップし、別の構文または指示文に進みます。たとえば、LOGICAL VOLUME LOOP を定義したが、システムに論理ボリュームがない場合、adviser スクリプトは LOGICAL VOLUME LOOP をスキップし、次の構文に進みます。

プラットフォームに存在しないループの場合はエラーが生成されます。

各インターバルで LOOP 文が繰り返されると、文で使用されるメトリックの値が変わります。たとえば、以下の LOOP 文はシステム上のアクティブなアプリケーションごとに 1 回 PRINT 文を実行するため、各アプリケーションの名前が出力されます。

## PRINT 文

PRINT 文を使用すると、収集しているデータを標準出力 (padv コマンド コンソール) に出力できます。メトリックまたは計算済みの変数を記録するのに、PRINT 文を使用できます。

構文:

```
PRINT printlist

PRINT Example

PRINT "The Application OTHER has a total CPU of ",


 other:app_cpu_total_util, "%"
```

この文が呼び出されると、以下のようなメッセージが padv コマンド コンソールに書き込まれます。

The Application OTHER has a total CPU of 89%

## SYMPTOM 文

構文:

```
SYMPTOM variable [TYPE = {CPU, DISK, MEMORY, NETWORK}]

RULE measurement {>, <, >=, <=, ==, !=} value PROB probability

[RULE measurement {>, <, >=, <=, ==, !=} value PROB probability]

 .

 .

 .
```

SYMPTOM キーワードおよび RULE キーワードは、SYMPTOM 文でのみ使用し、別の構文では使用できません。SYMPTOM 文は、トップレベルの文でなければならず、他の文内にネストできません。

variable は、この兆候の名前になる変数名です。SYMPTOM 文で定義された変数名は別の構文で使用できますが、それらの構文で変数値を変更しないようにしてください。

RULE は SYMPTOM 文のオプションであり、独立して使用できません。RULE オプションは SYMPTOM 文内で必要な数だけ使用できます。

SYMPTOM 変数は、各インターバルで RULE に従って評価されます。

measurement は、RULE の一部として評価される変数またはメトリックの名前です。

value は、measurement と比較される定数、変数、メトリックです。

probability は数値定数、変数、メトリックです。

真であるすべての SYMPTOM RULE の確率を加算して、SYMPTOM 値を作成します。SYMPTOM 値は `padv` コマンド コンソールのメッセージ内に表示されます。

測定値と値との間の条件が真であるすべての確率の合計が、その兆候が起きている確率になります。

# Windows での Performance Collection Component の使用

Performance Collection Component グラフィカル インターフェイスにアクセスするには、以下のフォルダの [HP Operations Agent Software] アイコンをクリックします。

**スタート→すべてのプログラム→HP→Operations Agent→ Performance Collection Component**

Performance Collection Component メイン ウィンドウ

この章では、Performance Collection Component グラフィカル インターフェイスを使用して実行する次の作業について説明します。

データ型とクラス

要約レベル

抽出またはエクスポートするデータの範囲

ログ ファイル データの抽出とログ ファイル データのエクスポート

ログ ファイル データのアーカイブ

ログ ファイルのサイズ変更

ログ ファイルの走査

ログ ファイルの分析

エクスポート テンプレートの構成

ユーザー オプションの構成

収集パラメータの構成

アラーム定義の構成

ステータス確認

パフォーマンス カウンタ収集の構築

データの抽出、エクスポート、アーカイブが伴う作業で Performance Collection Component を使用し始める前に、以下の各項をお読みください。これらの項では、データ型と要約レベルの選択方法や、抽出、エクスポート、アーカイブの対象データの範囲の選択方法について説明します。

# データ型とクラス

抽出またはエクスポートの対象として、次のデータ型の scopent ログファイルデータを選択できます。

| データ型 | 測定タイプ |
|---|---|
| global | システム全体の情報、つまりグローバル情報 |
| application | ユーザー定義の各アプリケーションに含まれるプロセス |
| configuration | システム構成の使用率 |
| process | 選択した対象プロセス |
| disk | ディスクデバイスの使用量 |
| filesystem | 論理デバイスの使用量 |
| logicalsystem | 論理システムの使用量 |
| cpu | CPU の使用量 |
| netif | ネットワークインターフェイスデバイスの使用量 |
| transaction | トランザクション追跡データ |

エクスポートする DSI ログファイルデータは、クラスに従って選択できます。各クラスは、1 つのソースの受信データを表し、一緒に記録される関連データ項目 (メトリック) のグループから構成されます。

# 要約レベル

データを抽出またはエクスポートするには (ログファイルデータを抽出またはエクスポートするときに)、希望の要約レベル (詳細、要約、またはその両方) を指定する必要があります。

詳細を指定すると、5 分間隔の詳細データが、プロセスデータを除くすべてのデータ型からエクスポートされます。プロセスデータからは、1 分間隔の詳細データがエクスポートされます。

要約を指定すると、1時間分の要約データがエクスポートされます。

詳細と要約ではエクスポート データの量が最大になります。

要約は、エクスポート データのサイズに影響します。たとえば、1 時間ごとの要約データは、5 分間の詳細データのサイズの約 10 分の 1 になります。

# 抽出またはエクスポートするデータの範囲

抽出またはエクスポートするデータは、データの記録日時に基づいて選択できます。たとえば、特定の日付から 30 日間にわたって毎日 (月曜日から日曜日まで) 8:00 am から 8:00 pm の間に記録されたデータを選択することが可能です。

エクスポートするデータの範囲を指定しないと、データは、デフォルトの開始日 (ログファイル内の最終日から 30 日前の日付) を使用して抽出またはエクスポートされます。存在するデータの量が 30 日分に満たない場合、ログファイル内で最も古いレコードの日付が開始日になります。

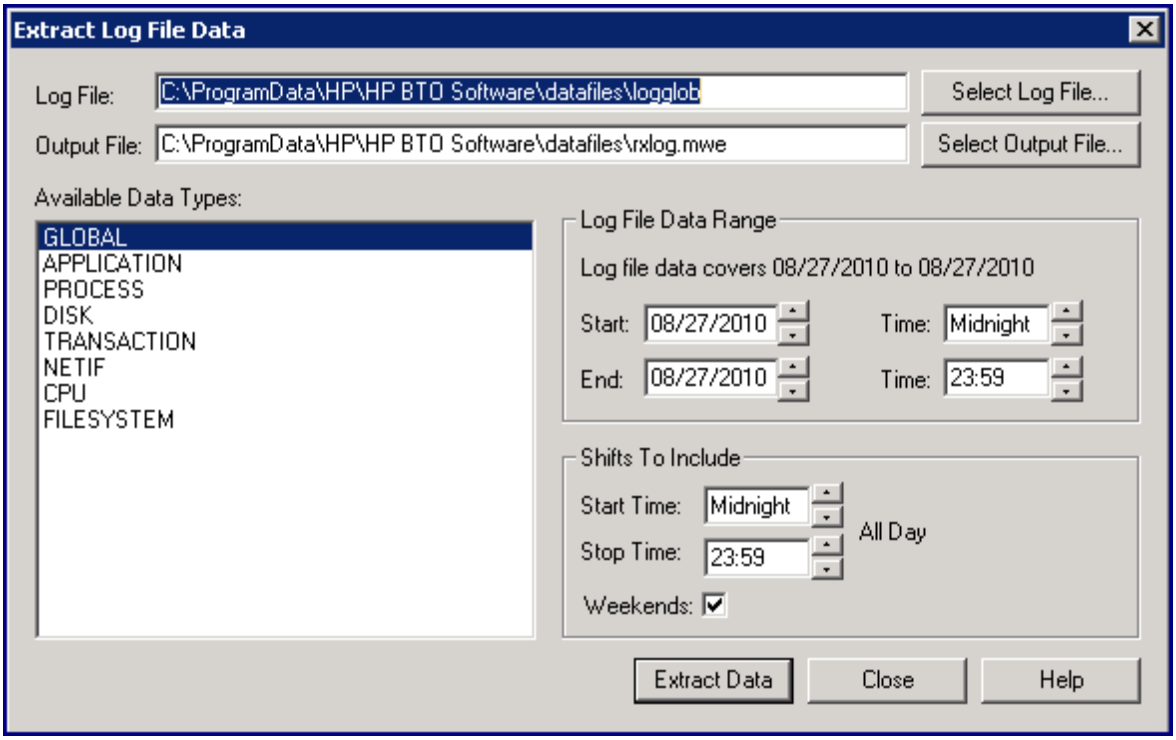
月曜日から金曜日までの作業シフトに対応する時間帯 (8:00 am から 5:00 pm までなど) に記録されたデータのみを、抽出またはエクスポートすることもできます。シフトを指定しないと、デフォルトとして、週末を含めた毎日の 24 時間分のデータが抽出またはエクスポートされます。

# ログ ファイル データの抽出

データコレクタである scopent は、データを継続的に収集し、それを生ログファイルに記録します。ユーザーは、デフォルトのグローバルログファイルセットである **logglob** から抽出ログファイルに特定のデータを抽出できます。抽出ログファイルは、後で Performance Manager などの分析プログラムによる分析やアーカイブに使用できます。また、既存の抽出ログファイルからもデータを抽出できます。DSI ログファイルデータは抽出できません。

**logglob** を指定すると、ログファイルセット内に含まれる他のすべての生ログファイルが自動的にオープンします。たとえば、プロセスデータを抽出するために **logproc** ログファイルをオープンする必要はありません。**logglob** をオープンすると、生ログファイルセット内のすべてのデータ型にアクセスできます。

[Extract Log File Data] ダイアログボックス




**注 :** Performance Collection Component を開始した後に [Extract Log File Data] または [Export Log File Data] ダイアログボックスを表示すると、現在アクティブなログファイルを示すためにデフォルトのグローバルログファイル名 **logglob** が [Log File] ボックスに表示されます。**Logglob**

は、Performance Collection Component を終了するか、別のログファイルを選択するまでアクティブです。別のログファイルを選択すると、そのファイルの名前が現在アクティブなログファイルとして [Log File] ボックスに表示されます。

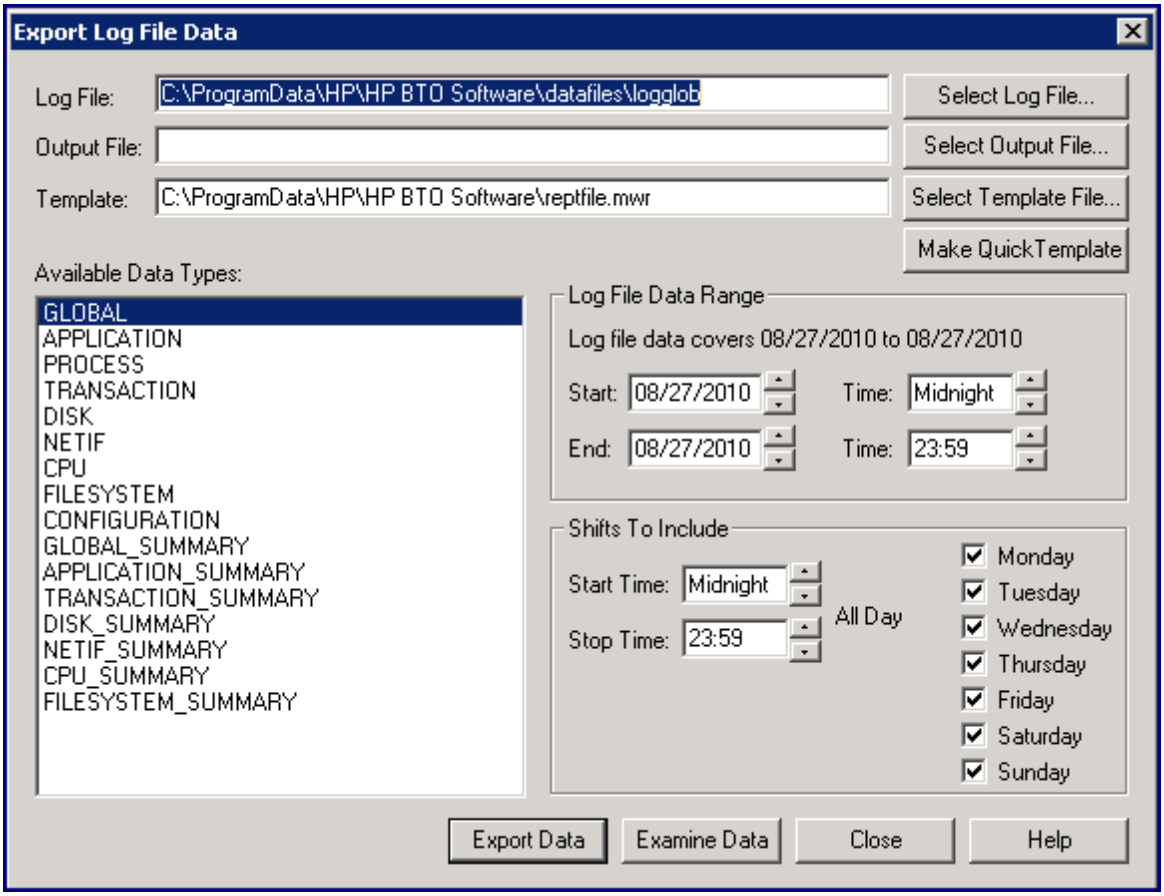
# ログ ファイル データのエクスポート

特定のデータを scopent の生ログファイルまたは抽出ログファイル、または DSI ログファイルから書式付きエクスポート ファイルに抽出できます。書式付きエクスポート ファイルは、表計算ソフトやその他のレポート作成ツールで使用できます。

エクスポート関数はデータをログファイルから削除しません。

以下の項では、次の事柄について説明します。

- エクスポート データに指定できるファイル属性
- エクスポート タスクを簡略化するためのエクスポート テンプレート
- デフォルトのエクスポート ファイル
- エクスポート タスクの概要
- [Export Log File Data] ダイアログ ボックス



# ファイル属 性

エクスポート データには、ファイルのフォーマット、欠 落 データを表 す値 、フィールド の区 切り記 号 、カラムのヘッダー、分 単 位 の要 約 インターバル、レイアウト、データ型 、エクスポート ファイルに含 めるメトリックなど、各 種 のファイル属 性 を割り当 てられます。これらの属 性 は、エクスポート テンプレート ファイルに保 存 できます。また、[Make Quick Template] ダイアログ ボックスを使 用 して直 接 指 定 できます(詳細 については、「クイック エクスポート テンプレート の作 成 」を参 照 してください)。

## ファイル フォーマット

エクスポート ファイルの出 力 フォーマットには、ASCII、データファイル、バイナリ、WK1 (表 計 算 シート) のいずれかを指 定 できます。

ASCII フォーマットは印 刷 可 能 な 文 字 データで、印 刷 されるレポートや、ユーザーが記 述 したプログラムまたはバッチ ファイルによるポスト処 理 に適 したものです。

datafile フォーマットは、数 値 以 外 の項 目 が二 重 引 用 符 (" ") で囲 まれる点 を除 いて ASCII と同じです。datafile フォーマットは、ほとんどの表 計 算 ソフトおよびグラフ作 成 ソフトへのデータ転 送 に使 用 できます。

binary フォーマットは印 刷 できません。数 値 が 2 進 整 数 値 として表 されるためデータがコンパクトになります。変 換 が最 小 限 で済 み、メトリック精 度 が最 高 であるため、このフォーマットは、ユーザーが記 述 した分 析 プログラムへの入 力 に最 も適 しています。

WK1 (表 計 算 シート) フォーマットは、Microsoft Excel などの表 計 算 ソフト、データベース、グラフ作 成 ソフトに対 応 します。

## 欠 落 値

エクスポート ファイルには、ソース ログ ファイルから欠 落 したデータを補 うデータ値 を入 れることができます。データの欠 落 は、特 定 のバージョンの scopent コレクタで使 用 できないメトリックがある場 合 に発 生 します。さらに、アプリケーション、ディスク、トランザクションに関 する複 数 レイアウト エクスポート フォーマットは、すべてのアプリケーション、ディスク、トランザクションに関 する出 力 レコード に領 域 を確 保 します。特 定 のエントリに関 してデータが一 定 期 間 記 録 されない場 合 、そのデータは「欠 落 データ」になります。

## フィールド 区 切 り記 号

ASCII フォーマットおよびデータファイル フォーマット のエクスポート ファイルでは、メトリック間 にフィールド 区 切 り記 号 を挿 入 できます。区 切 り記 号 として使 用 する文 字 には、印 刷 可 能 なものと印 刷 不 可 能 なもの (タブ文 字 など) があります。

デフォルト の区 切 り記 号 はスペースですが、多 くのプログラムでは、フィールド 区 切 り記 号 としてカンマを使 用 する必 要 があります。

## 分単位の要約インターバル

各要約インターバルを分単位で指定します。各要約インターバルは、分単位で 5～ 1440 分 (1 日) の範囲で指定できます。

## ヘッダー

エクスポート ファイルにはカラムのヘッダーを含められます。ファイル内の最初のレコードはエクスポートデータです (WK1 フォーマットのファイルを除く)。ただし、ファイルにヘッダーを含める場合、ASCII フォーマットおよび datafile フォーマットのファイルでは、エクスポート ファイルのタイトル (指定されている場合) とメトリックの各カラムのヘッダーがファイルの最初のレコードの前に書き込まれます。バイナリフォーマット ファイルでは、ヘッダーがファイルの最初のレコードの前に書き込まれ、メトリックの説明が含まれます。

WK1 ファイルでは、必ずヘッダーが書き込まれます。

## 複数レイアウト

アプリケーションやディスクなどのマルチインスタンスのデータ型には、(1 つのレコード出力に) 複数のレイアウトを指定できます。

単一のレイアウトは、特定の時間インターバルでアクティブであった各アプリケーションまたは各ディスクについて 1 レコードを書き込みます。複数のレイアウトは、各時間インターバルで 1 レコードを書き込みます。このレコードの一部は、構成されている各アプリケーションまたは各ディスクについて保存されます。

## エクスポート ファイルのタイトル

エクスポート ファイルにはタイトルを指定できます。タイトルには、リテラル文字列と置換キーワードを使用できます。export title 文字列には、次の置換キーワードが使用できます。

| | |
|---|---|
| `!date` | エクスポート ファイルの作成日 |
| `!time` | エクスポート ファイルの作成時刻 |
| `!logfile` | データ取得元のログ ファイルの名前 |
| `!collector` | コレクタ プログラム (scopent または dsilog) の名前とバージョン |
| `!class` | 必要なデータの型 |
| `!system_id` | scopent の生ログ ファイル データまたは抽出ログ ファイル データを収集するシステムの識別子 (DSI ログ ファイル データに対しては無効) |

たとえば、次のような文字列を入力します。

```
export "!system_id data from !logfile on !date !time"
```

この文 字 列 により、次 のようなタイト ルが生 成 されます。

```
gemini data from logglob on 10/25/99 08:30 AM
```

# エクスポート ファイルのテンプレート

エクスポート タスクは、エクスポート ファイルのファイル属 性 を定 義 するエクスポート テンプレートを使 用 します。デフォルト のファイル属 性 は、次 の指 定 が含 まれるファイル **<rpmtools>\data\reptfile.mwr** から得 られます。

ASCII ファイル フォーマット

欠 落 値 に対 する 0 (ゼロ)

フィールド 区 切 り記 号 の空 白

60 分 ごとの要 約

ヘッダーを含 める

特 定 のデータ型 に推 奨 されるメトリック セットをエクスポートに含 める

別 のエクスポート テンプレート ファイルを指 定 しても、特 別 なファイル属 性 を指 定 してもかまいません (この章 で後 述 している「クイック エクスポート テンプレート の作 成 」を参 照 してください)。

カスタマイズしたエクスポート テンプレートも [Configure Export Templates] ダイアログ ボックスを使 用 して作 成 できます (この章 で後 述 している「エクスポート テンプレート の構 成 」を参 照 してください)。

# デフォルト のエクスポート ファイル

エクスポート データを収 める出 力 ファイルを指 定 しないと、エクスポート タスクでは、指 定 したデータ型 および要 約 レベルに基 づいて **<*ディスクドライブ*>:\Program Files\HP\HP BTO Software\data\datafiles** ディレクトリにデフォルト の出 力 ファイルを作 成 します。

## scopent データ

scopent ログ ファイル データをエクスポート する場 合 、次 のデフォルト ファイル名 がエクスポート ファイル に割 り当 てられます。

| | |
|---|---|
| **xfrdGLOBAL.ext** | グローバル詳 細 データ |
| **xfrsGLOBAL.ext** | グローバル要 約 データ |
| **xfrdAPPLICATION.ext** | アプリケーション詳 細 データ |
| **xfrsAPPLICATION.ext** | アプリケーション要 約 データ |
| **xfrdPROCESS.ext** | プロセス詳 細 データ |
| **xfrdDISK.ext** | ディスク デバイス詳 細 データ |
| **xfrsDISK.ext** | ディスク デバイス要 約 データ |

| | |
|---|---|
| **xfrdCPU.ext** | CPU 詳細データ |
| **xfrsCPU.ext** | CPU 要約データ |
| **xfrdFILESYSTEM.ext** | filesystem 詳細データ |
| **xfrsFILESYSTEM.ext** | filesystem 要約データ |
| **xfrdNETIF.ext** | netif 詳細データ |
| **xfrsNETIF.ext** | netif 要約データ |
| **xfrdTRANSACTION.ext** | トランザクション追跡詳細データ |
| **xfrsTRANSACTION.ext** | トランザクション追跡要約データ |
| **xfrdCONFIGURATION.ext** | 構成詳細データ |
| xfrdLOGICAL.ext | 論理システム詳細データファイル |
| xfrsLOGICAL.ext | 論理システム要約データファイル |

エクスポート タスクが完了すると、[Export Log File Data] ダイアログ ボックスの [**Examine Data**] ボタンをクリックして、出力エクスポート ファイルの内容を表示できます。

デフォルトのファイル名は、データ型名から作成されます。データが詳細データであるか要約データであるかによって、接頭辞が **xfrd** または **xfrs** になります。拡張子 (**.ext**) は、エクスポート テンプレート ファイルで指定されているファイル フォーマットに応じて、**asc** (ASCII)、**bin** (バイナリ)、**dat** (データファイル)、**wk1** (表計算シート) のいずれかになります。

たとえば、次のようになります。

**xfrdNETIF.wk1** には、NETIF データ型に関する詳細データが表計算シート フォーマットで書き込まれています。

**xfrsAPPLICATION.asc** には、アプリケーション データ型に関する要約データが ASCII フォーマットで書き込まれています。

## DSI データ

DSI ログ ファイル データを抽出する場合、デフォルトのファイル名はクラス名から作成されます。データが詳細データであるか要約データであるかによって、接頭辞が **xfrd** または **xfrs** になります。拡張子は、エクスポート テンプレート ファイルで指定されているファイル フォーマットに応じて、**asc** (ASCII)、**dat** (データファイル)、**wk1** (表計算シート) のいずれかです。

たとえば、次のようになります。

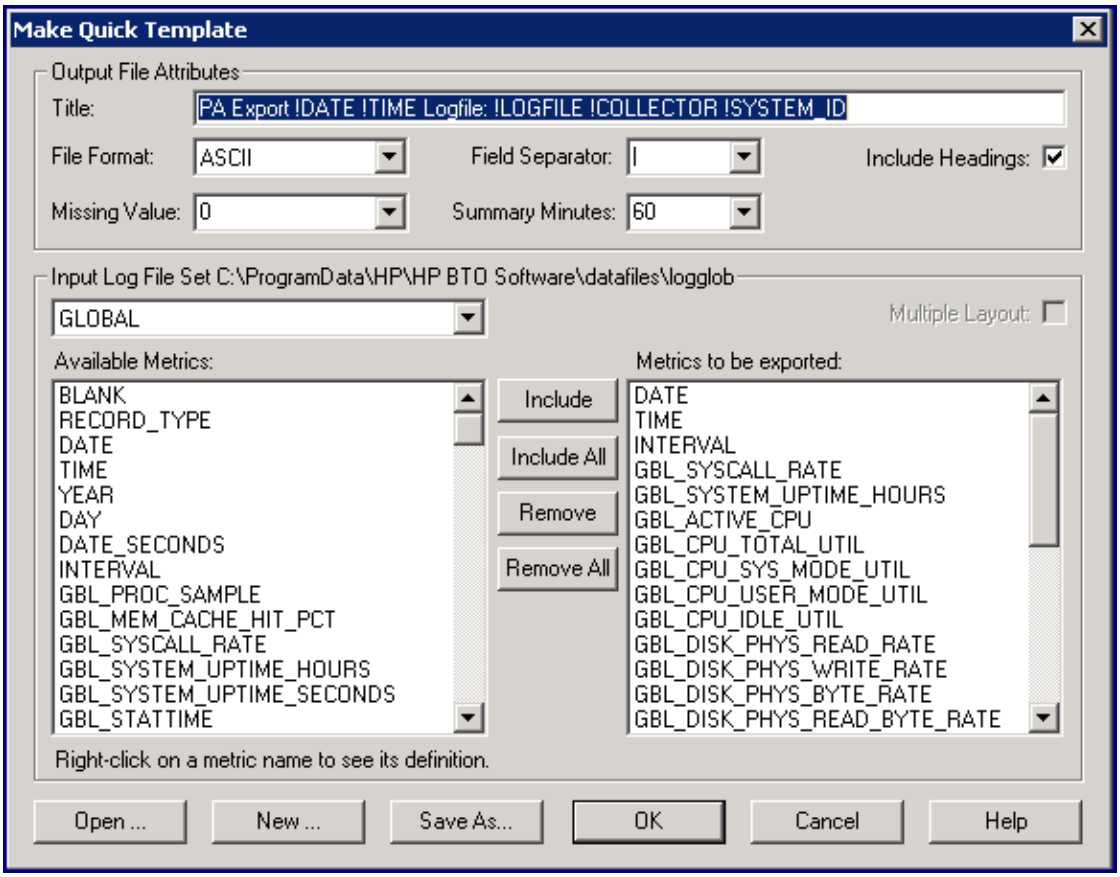
**xfrdACCTG.wk1** には、ACCTG クラスに関する詳細データが表計算シート フォーマットで書き込まれています。

**xfrsPERSONL.asc** には、PERSONL クラスに関する要約データが ASCII フォーマットで書き込まれています。

# クイック エクスポート テンプレートの作成

エクスポート ファイルに含めるメトリックを変更するときや、エクスポート用に選択したエクスポート テンプレートに指定されているファイル属性やメトリックを変更するときは、[Export Log File Data] ダイアログ ボックスの [Make Quick Template] 機能を使用します。

[Make Quick Template] ダイアログ ボックス




クイック テンプレートを削除する手順は以下のとおりです。

[Export Log File Data] ダイアログ ボックスの [**Make Quick Template**] ボタンをクリックします。[Make Quick Template] ダイアログ ボックスが表示され、エクスポート用に選択しているエクスポート ファイルのタイトルが示されます。

[Output File Attributes] の下にある各ボックスには、エクスポート用に選択しているエクスポート テンプレートのファイル属性の設定に基づき、エクスポート ファイルに関する現在の設定値が示されます。これらの設定値は変更できます。

別のエクスポート テンプレートファイルを使用するには、[**Open**] ボタンをクリックします。

[Make Quick Template] ダイアログ ボックスから既存の設定値をすべてクリアし、[**New**] ボタンをクリックしてまったく新しいエクスポート テンプレート ファイルを作成することもできます。

# エクスポートするメトリックの選択

エクスポートするメトリックのデータ型またはクラスを選択した後、エクスポートに含めるメトリックを選択できます。それぞれのデータ型またはクラスには独自のメトリック セットがあり、[Available Metrics] にリストされます。[Metrics to be exported] にリストされるメトリックは、エクスポートに含める現在のエクスポート テンプレートで指定されているメトリックです。このリストはそのまま使用できますが、リストからメトリックを削除したり、他のメトリックを選択してエクスポートに含めることもできます。

アプリケーション、ディスク、cpu、ファイルシステム、netif またはトランザクション のいずれかのデータ型を選択した場合、[**Multiple Layout**] チェック ボックスを選択すると複数レイアウト (1 レコード出力につき) を生成できます。また、[**Multiple Layout**] チェック ボックスを選択しないと単一レイアウトを生成できます。

# 選択内容の保存

選択後に、次のいずれかの処理が行えます。

テンプレート ファイルに対する変更内容を保存せずに、エクスポートするファイル属性およびメトリックのリストに対して行った選択を使用して、エクスポートを続行できます。

将来のエクスポートで使用できるように、新しいエクスポート テンプレート ファイルに選択内容を保存できます。元のエクスポート テンプレート ファイルは変更せずにそのままにします。

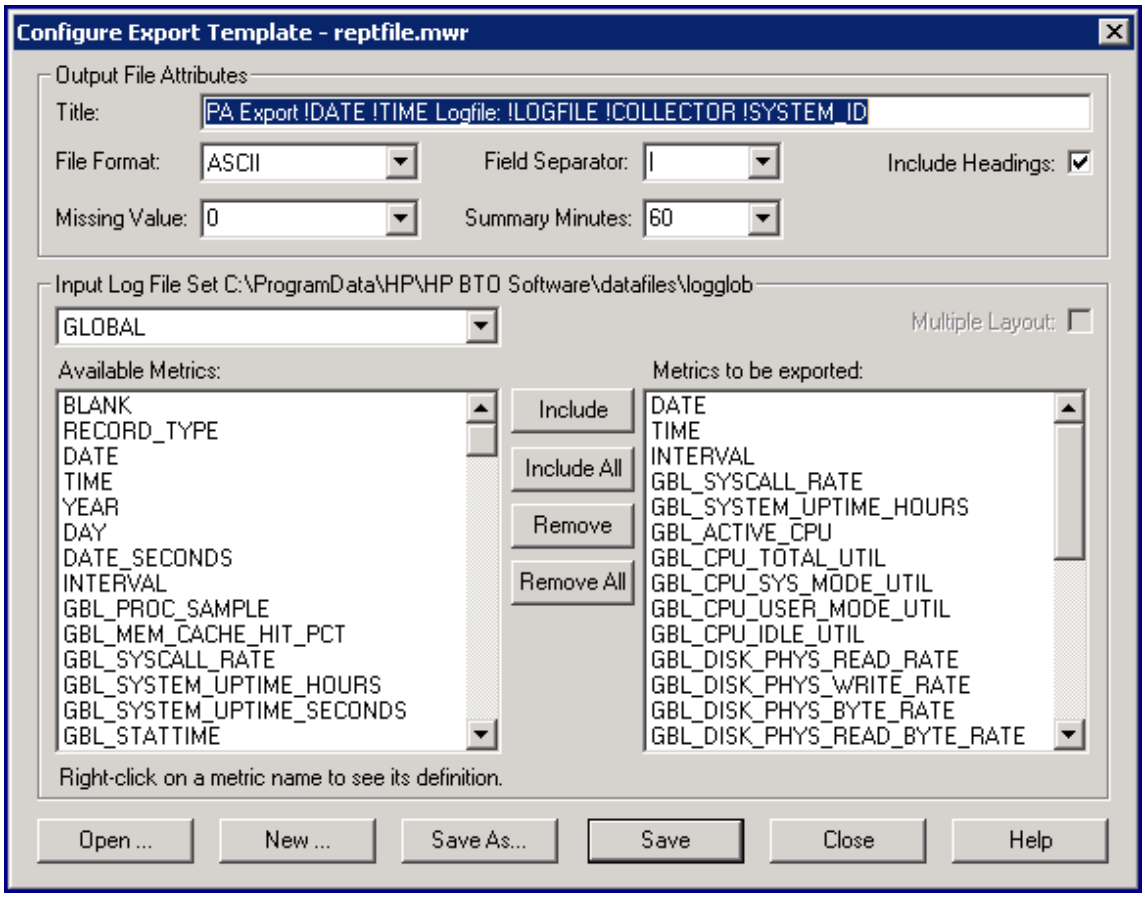
> **注:** 選択内容を新しいエクスポート テンプレート ファイルに保存する場合、新しいテンプレート ファイル名を指定するときにファイル名拡張子 **.mwr** を含める必要があります。たとえば、**mytmplte.mwr** のようなファイル名です。

クイック エクスポートのテンプレートの作成手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I…?**] を選んでから、[**Make a quick export template**] を選んでください。

# エクスポート テンプレートの構成

既存のエクスポート テンプレート ファイルをカスタマイズするときや、エクスポート テンプレート ファイルを新規作成するときには、[Configure] メニューの [**Export Templates**] コマンドを使用します。テンプレートに含める新しいファイル属性やメトリックを選択するときには、[Configure Export Template] ダイアログ ボックスを使用します。

[Configure Export Template] ダイアログ ボックス




エクスポート テンプレートを設定するには、次の手順を実行します。

メイン ウィンドウの [Configure] メニューから [**Export Templates**] をクリックします。[Configure Export Template] ダイアログ ボックスが表示され、現在オープンしているエクスポート テンプレート ファイルの名前がダイアログ ボックス タイトルに示されます。別のエクスポート テンプレートファイルを編集するには、[**Open**] ボタンをクリックします。

[Output File Attributes] の下にある各ボックスには、構成しているエクスポート テンプレートのファイル属性の設定に基づき、エクスポート ファイルに関する現在の設定値が示されます。これらの設定値は変更できます。

[Configure Export Template] ダイアログ ボックスから既存の設定値をすべてクリアし、[**New**] ボタンをクリックしてまったく新しいエクスポート テンプレート ファイルを作成することもできます。

## エクスポートするメトリックの選択

エクスポートするメトリックのデータ型またはクラスを選択した後、エクスポートに含めるメトリックを選択できます。それぞれのデータ型またはクラスには独自のメトリック セットがあり、[Available Metrics] にリストされます。[Metrics to be exported] にリストされるメトリックは、エクスポートに含める現在のエクスポート テンプレートで指定されているメトリックです。このリストはそのまま使用できますが、リストからメトリックを削除したり、他のメトリックを選択してエクスポートに含めることもできます。

アプリケーション、ディスク、cpu、ファイルシステム、netif またはトランザクション のいずれかのデータ型を選択した場合、[**Multiple Layout**] チェック ボックスを選択すると複数レイアウト (1 レコード出力につ

き) を生 成 できます。また、[**Multiple Layout**] チェック ボックスを選 択しないと単 一 レイアウトを生 成 できます。

## 選 択 内 容 の保 存

編 集したテンプレートを保 存 するときには、次 の 3 つの選 択 肢 があります。

現 在 のテンプレート ファイルに変 更 内 容 を保 存 します。

新しいテンプレート ファイルに変 更 内 容 を保 存 します。この場 合 、元 のテンプレート ファイルは変 更 されません。

変 更 内 容 をキャンセルして、テンプレート ファイルの変 更 を防 ぎます。

| | |
|---|---|
| | 選 択 内 容 を新しいエクスポート テンプレート ファイルに保 存 する場 合 、新しいファイル名 を指 定 するときにファイル名 拡 張 子 .mwr を含 める必 要 があります。<br><br>テンプレート ファイルに変 更 を加 え、変 更 内 容 を保 存 する前 に [**Close**] ボタンをクリックすると、変 更 内 容 のキャンセルまたは保 存 を促 すプロンプトが表 示 されます。[**Cancel**] ボタンをクリックすると、[Configure Export Template] ダイアログ ボックスに戻 ります。 |

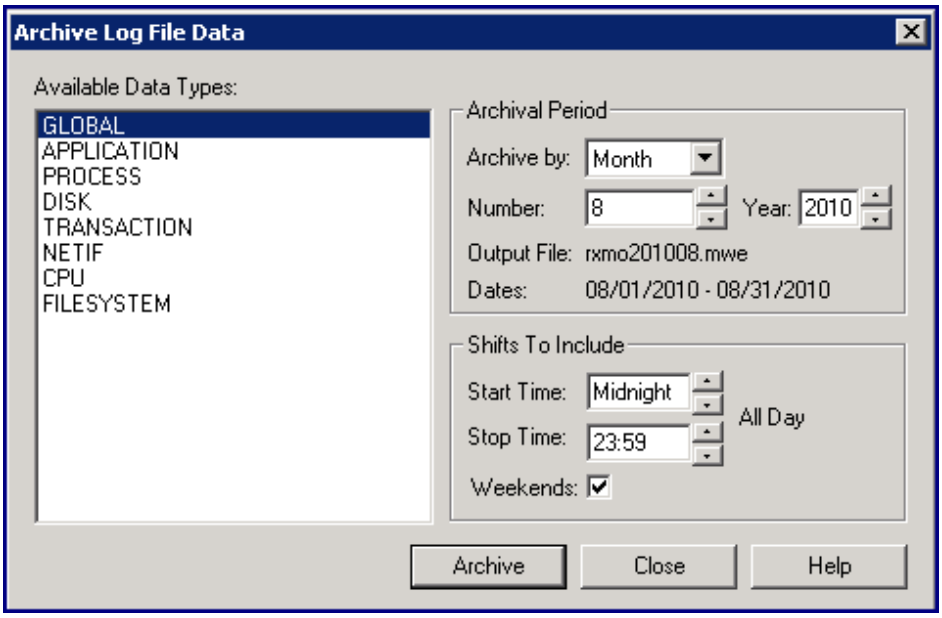
エクスポート テンプレート ファイルの構 成 手 順 を参 照 するには、[Help] メニューから [**Help Topics**] を選 択 し、[**How Do I…?**] を選 んでから、[**Configure an export template file**] を選 んでください。

## ログ ファイル データのアーカイブ

アーカイブや将 来 のデータ分 析 に使 用 する scopent ログ ファイル データの選 択 箇 所 を抽 出 するときには、[Logfile] メニューの [**Archive**] コマンドを使 用 します。

アーカイブが目 的 の場 合 、データは生 ログ ファイルから抽 出 できます。
抽 出 データは、**<***ディスクドライブ***>:Program Files\HP\HP BTO Software\data\datafiles** ディレクトリにあるアーカイブ出 力 ファイルに自 動 的 に配 置 されます。このディレクトリ名 は、選 択 しているアーカイブ期 間 を反 映 します。このようなファイルは、テープにコピーしてオフラインで保 存 した後 、ディスク領 域 を解 放 するために削 除 できます。

[Archive Log File Data] ダイアログ ボックス




## アーカイブ期間

特定の週、月、年に記録されたデータに基づき選択できます。

また、作業シフトに対応する特定の時間帯に記録されたデータのみを抽出したり、週末 (土曜日と日曜日) を含めたり除外したりできます。シフトを指定しないと、デフォルトとして毎日の 24 時間分のデータが抽出されます。デフォルトでは、週末が含まれます。

## アーカイブ データの追加

アーカイブ関数は特殊な機能を備えています。選択したアーカイブ期間 (週間、月間、年間) に応じて、前回のアーカイブ期間の出力ファイルに最終日までの抽出データが含まれているかどうかが自動的に確認されます。最終日までの抽出データが含まれていない場合、その出力ファイルにデータが追加され、前回のアーカイブ期間の抽出が完了します。

たとえば、1999 年 5 月 7 日に、1999 年 5 月の月間データのアーカイブを開始すると仮定します。出力ファイル **rxmo199905.mwe** が作成され、5 月 1 日から現在の日付 (5 月 7 日) までのデータが収められます。

1999 年 6 月 4 日に、次の月間アーカイブ期間が呼び出されます。6 月の **rxmo199906.mwe** ファイルが作成される前に、前月の **rxmo199905.mwe** ファイルが確認されます。このファイルが完成していない場合は、データが追加され、1999 年 5 月 31 日までの抽出を完了します。次に、**rxmo199906.mwe** ファイルが作成され、1999 年 6 月 1 日から現在の日付 (6 月 4 日) までのデータを保持します。

別の月間 (週間、年間) アーカイブが月 (週、年) に少なくとも 1 度呼び出されると、この機能は各アーカイブ期間のファイルを完成させてから次のアーカイブ期間のファイルを作成します。

## アーカイブに関するヒント

ログファイル データのアーカイブに関するヒントを次に示します。

月に1度、月間アーカイブ期間を指定し、各生ログファイルからすべての詳細データを1つの抽出ログファイルに抽出してください。

毎月 64 メガバイトを超えるデータがシステムで生成される場合、データを毎週抽出することをお勧めします。また、プロセス詳細データを抽出から除外することもできます。

グローバル要約データとアプリケーション要約データを年単位で抽出すると、必要なディスク領域を最小限に抑えられます。また、このアーカイブファイルは、長期的な傾向変動分析に使用できます。

ログファイルデータをアーカイブするには、次の手順を実行します。

メインウィンドウの [Logfile] メニューから [**Archive**] をクリックします。[Archive Log File Data] ダイアログボックスが表示されます。アーカイブするデータは、生ログファイルセットから抽出されます。

アーカイブするデータ型を [Available Data Types] 一覧から選択してから、アーカイブ期間 (Week、Month、Year のいずれか) と、対象シフトを指定します。

[**Archive**] ボタンをクリックして、アーカイブ処理を開始します。

ログファイルデータのアーカイブ手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Archive log file data**] を選んでください。

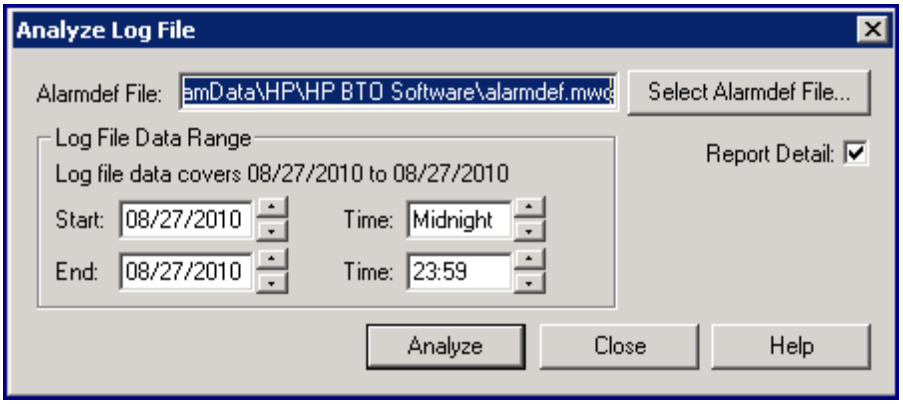
# ログファイルの分析

アラーム定義ファイルのアラーム定義や実際に発生したアラームの活動状況に関するレポートと比較して生ログファイルセットのデータを分析するときには、[Logfile] メニューの [**Analyze Log File**] コマンドを使用します。

このタスクを実行すると、アラーム定義がシステムで収集された履歴データと適合しているかどうか評価できます。また、アラーム定義が分析用システム上で非常に多数のアラームを発生させるか、少数のアラームを発生させるかを決定できます。

分析する生ログファイルは、Performance Collection Component のデフォルト データソースである SCOPE で参照します。別のログファイルを分析するには、対象のログファイルを参照するデータソースの名前を指定する USE 文をアラーム定義に配置する必要があります。

[Analyze Log File] ダイアログボックス



# 分析するデータの範囲

特定期間に収集されたログファイルデータを分析できます。分析するデータの範囲を指定しないと、データは、デフォルトの開始日 (ログファイル内の最終日から30日前の日付。30日以前のデータがない場合は、ログファイル内の最新レコードの日付)を使用して分析されます。

# 分析レポート

この作業を行うと、複数のアラームイベントと1つのアラーム要約をリストした印刷可能なレポートが生成されます(アラームイベントは、[Analyze Log File] ダイアログボックスの [Report Detail] ボックスがチェックされているときにのみリストされます)。

アラームイベントには、START、END、REPEAT の各アラームのステータスと、関連する PRINT 文のテキストが含まれます。また、PRINT 文のテキストが条件としてリストされている場合 (IF 文で)、その条件が真になるとテキストが含まれます。EXEC 文は実行されませんが、リストされるため、実行内容を参照できます。

アラーム要約には、アラームの発生回数と、各アラームがアクティブな状態 (オン) であった時間が示されます。回数には、アラームの開始回数と繰り返し回数が含まれますが、アラームの終了回数は含まれません。

ログファイルを分析する手順は以下のとおりです。

1. メインウィンドウの [Logfile] メニューから [**Analyze**] をクリックします。[**Analyze Log File**] ダイアログボックスが表示され、現在選択しているアラーム定義ファイルの名前が示されます。
2. 別のアラーム定義ファイルを使用するには、[**Select Alarmdef File**] ボタンをクリックします。
3. 分析するログファイルデータの範囲を選択します。
4. 分析レポートにアラームイベントを含めるには、[**Report Detail**] ボックスをチェックします。チェックしない場合、アラーム要約のみが生成されます。
5. [**Analyze**] ボタンをクリックして分析を開始します。分析結果は [MeasureWare Agent Report Viewer] ウィンドウに表示されます。

ログファイルの分析手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I…?**] を選んでから、[**Analyze a log file**] を選んでください。

# ログファイルの走査

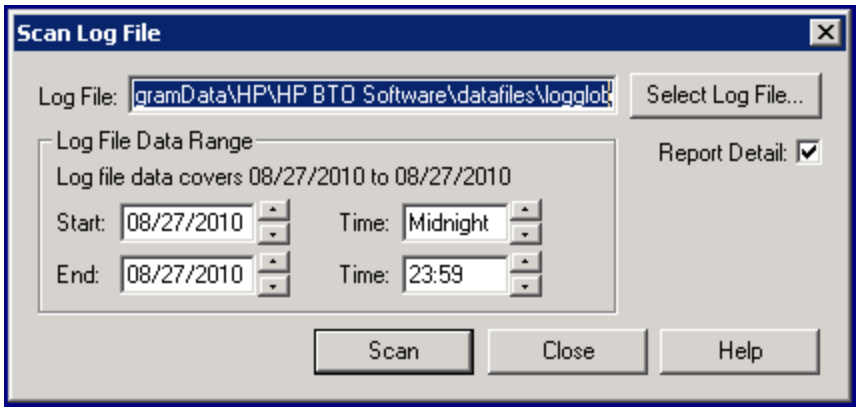
scopent ログファイルを走査してその内容に関するレポートを作成するときは、[Logfile] メニューの [**Scan Log File**] コマンドを使用します。特定期間に収集されたデータを検索するためにログファイル全体、およびログファイルの一部を走査できます。

走査の結果生成されるレポートは、12 のセクションから構成されます。次の 4 つのセクションが必ず出力されます。

- プロセス要約レポート
- コレクタの適用範囲の要約
- ログファイルの内容の要約
- ログファイルの空きスペースの要約

次の8つのセクションは、[Scan Log File] ダイアログ ボックスで [**Report Detail**] を選択したときにのみ出力されます。

- 初期 **parm** ファイルのグローバル情報およびシステム構成情報
- 初期 **parm** ファイルのアプリケーション定義
- **parm** ファイルのグローバル変更内容
- **parm** ファイルのアプリケーションおよび変更通知
- コレクタのオフタイム通知
- アプリケーション固有の要約レポート




ログファイルをスキャンする手順は以下のとおりです。

1. メイン ウィンドウの [Logfile] メニューから [**Scan Log File**] をクリックします。[Scan Log File] ダイアログ ボックスが表示され、現在オープンしているログファイルの名前が反転表示されます。
2. 別のログファイルを走査するには、[**Select Log File**] ボタンをクリックします。
3. 特定期間に記録されたデータを走査するには、[Log File Data Range] でその期間の開始日時および終了日時を選択するか入力してください。
4. 完全な走査レポートが必要な場合、[**Report Detail**] ボックスをチェックします。チェックしない場合、レポートのサブセットのみが生成されます。
5. 走査プロセスを開始するには、[**Scan**] ボタンをクリックします。走査結果は [Performance Collection Component Report Viewer] ウィンドウに表示されます。
6. ログファイル データの走査手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Scan a log file**] を選んでください。

# ログ ファイルのサイズ変更

scopent の生ログファイルのサイズを変更するときには、[Logfile] メニューの [**Resize Log File**] コマンドを使用します。サイズ変更は、特定のファイルのメガバイト単位のサイズを使用するか、ファイルにデータを保持する日数を使用して行います。

生ログファイルの最大サイズは **parm** ファイルの size パラメータで指定します。ログファイルのサイズを変更すると、ログファイルデータのロールバック頻度を制御しやすくなります。

サイズを変更するデータ型 (グローバル、アプリケーション、プロセス、デバイス、トランザクションのいずれか) は選択できます。各データ型は、**logglob**、**logappl**、**logproc**、**logdev**、**logtran** の各生ログファイルに対応しています。次に、サイズ変更の実行方法 (メガバイト数の変更または日数による変更) を選択します。選択したサイズ変更方法に応じて、[Resize Log File] ダイアログボックスの [Log File Settings] ボックスには、次のようなフィールドが示されます。

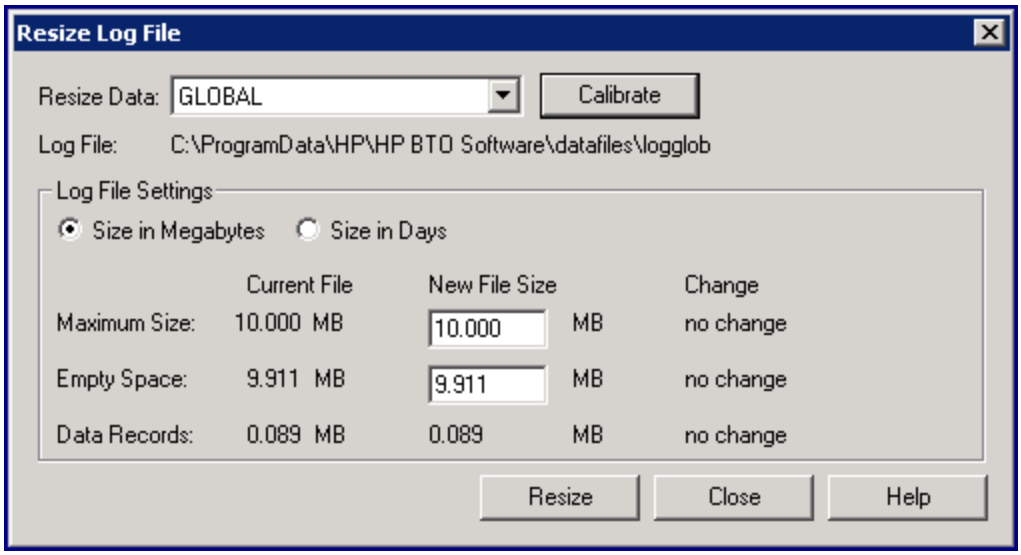
[Maximum Size] 各フィールドには、現在のファイルサイズ、新しいファイルサイズ、サイズ変更内容が示されます。

[Empty Space] の各フィールドには、現在のファイルの空き容量、サイズ変更プロセスの完了後にファイルに必要な容量、変更内容が示されます。これらの値は、サイズ変更プロセスで削除する必要があるデータがログファイル内にあるかどうかを判断するときに使用します。

[Data Records] の各フィールドには、現在のログファイルに含まれているデータレコードの量と、サイズ変更後にログファイルに含まれるデータレコードの量が示されます。

ログファイルのサイズはメガバイト単位で指定します。メガバイトではなく日数でサイズを指定する方が便利な場合もあります。[Size in Days] を選択すると、ダイアログボックスのすべての単位が「days」に変わります。メガバイトから日数への換算は、各データ型の「1 日あたりのメガバイト数」に基づいて行われます。この換算には、最初は推定値が使用されます。

より正確な値が、[**Calibrate**] ボタンをクリックすることで得られます。この補正関数は、既存のログファイルを実際に測定して、より正確な 1 日あたりのメガバイト数を算出します。メガバイト単位でサイズを指定している場合、換算は不要であるため、補正関数を使用する必要はありません。




ログファイルのサイズを変更する前に、scopent コレクタを終了する必要があります。scopent を停止するには、「データ収集の終了と再開始」の手順を実行します。

scopent を終了せずにログファイルのサイズ変更を行っても、既存のログファイルのサイズは変更されません。ログファイルのサイズを変更する手順は以下のとおりです。

scopent を終了してから、メインウィンドウの [Logfile] メニューから [**Resize Log File**] を選択して、[Resize Log File] ダイアログボックスを表示します。

[Resize Data] ボックスで、サイズを変更するデータ型 (グローバル、アプリケーション、プロセス、デバイス、トランザクションのいずれか) を選択します。

[**Size in Megabytes**] か [**Size in Days**] を選択します。選択した指定に応じて、[Current File] および [New File Size] が表示されます。

[New File Size] の表示に基づいてサイズ変更を行うには、[**Resize**] ボタンをクリックしてサイズ変更プロセスを開始します。

日数でサイズを決定するときにログファイルの追加容量をより正確に見積もるには、以下の手順を実行します。

[**Calibrate**] ボタンをクリックします。過去 30 日間にファイルに記録されたデータレコードの実数およびサイズが直ちに表示されます。

[**Close**] ボタンをクリックして、[Resize Log File] ダイアログボックスに戻ります。日数でサイズを決定する場合、補正に基づいて算出された[New Current File] および [New File Size] の値が表示されます。

[**Resize**] ボタンをクリックしてログファイルのサイズを変更します。

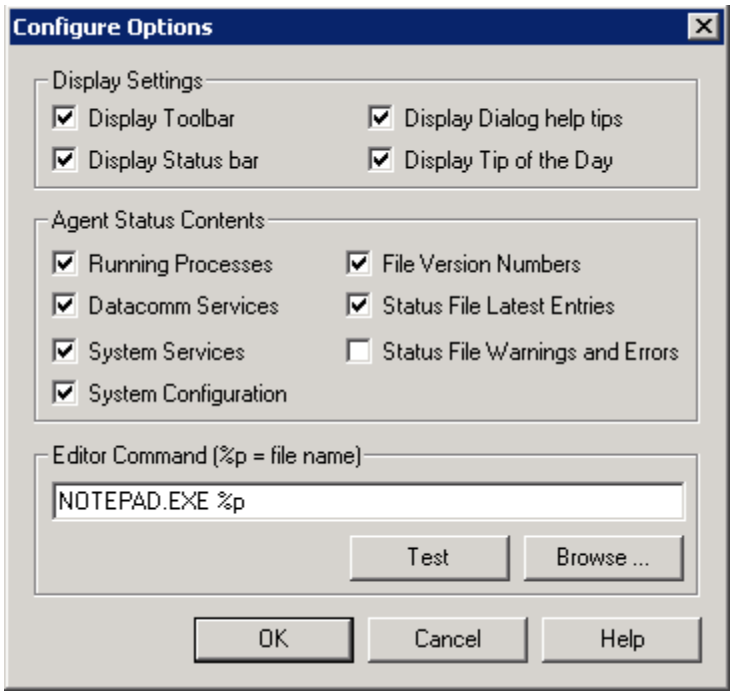
別のタスクを実行する前に、「データ収集の終了と再開始」の手順を使用して scopent を開始します。

ログファイルのサイズ変更手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I…?**] を選んでから、[**Resize a log file**] を選んでください。

# ユーザー オプションの構成

Performance Collection Component の使用中またはメインウィンドウ上で、ツールバー、ステータスバー、ダイアログヘルプのヒント、[Tip of the Day] の表示を制御するときには、[Configure] メニューの [**Options**] コマンドを使用します。

また、[**Options**] コマンドを使用して、収集パラメータやアラーム定義ファイルを変更するエディタやワード プロセッサを構成したり、[Agent] メニューの [**Status**] コマンドを選択するときに表示するステータス情報の種類を選択できます。




ユーザー オプションを設 定 するには、次 の手 順 を実 行 します。

1. メイン ウィンド ウの [Configure] メニューから [**Options**] コマンド をクリックして、[Configure Options] ダイアログ ボックスを表 示 します。
2. [**Display Toolbar**] チェック ボックスを選 択 して、メイン ウィンド ウにツールバーを表 示 します。
3. [**Display Status Bar**] チェック ボックスを選 択 して、各 ダイアログ ボックスの下 部 とメイン ウィンド ウに現 在 のステータスを表 示 します。
4. [**Display Dialog Help Tips**] チェック ボックスを選 択 して、ダイアログ ボックスにヘルプのヒントを表 示 します。
5. [Display Tip of the Day] チェック ボックスを選 択 して、[Performance Collection Component] メイン ウィンド ウをオープンしたときにその日 のヒントを表 示 します。

設 定 するには、以 下 の手 順 に従 います。

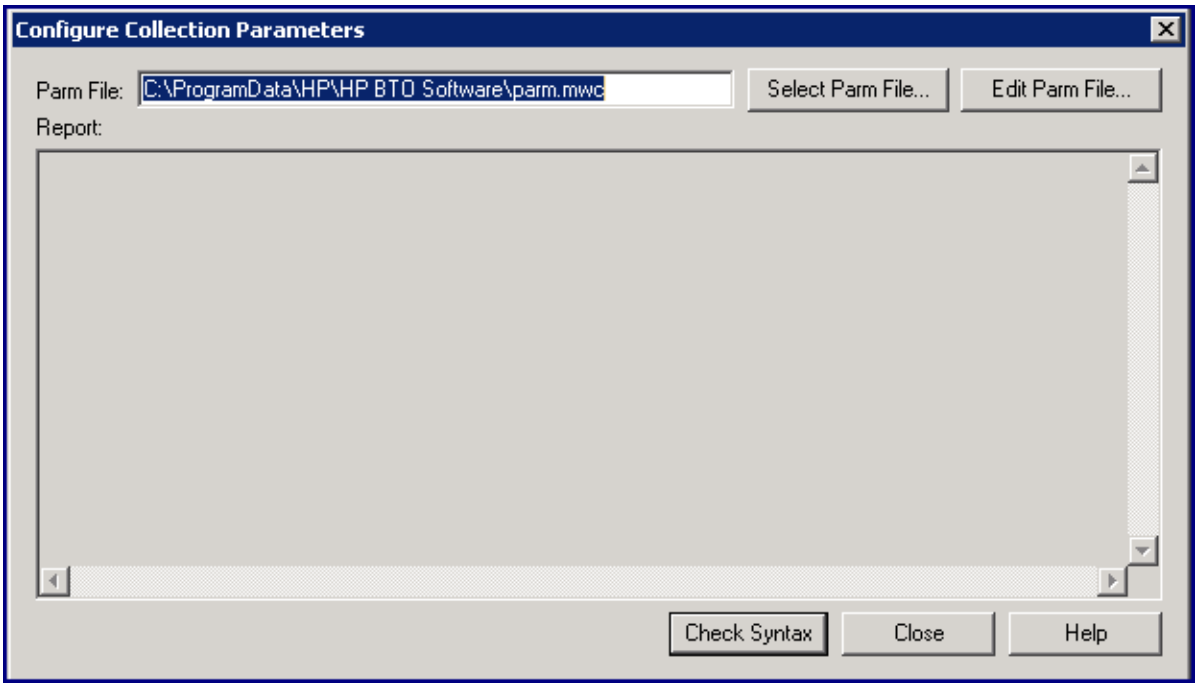
1. ファイル名 拡 張 子 `.exe` を使 用 してエディタのディレクトリ パスおよびファイル名 を [Editor Command] ボックスに入 力 します (`C:\MSOffice\winword\winword.exe` など)。
2. [**Browse**] ボタンをクリックして、[Select a Text Editor] ダイアログ ボックスを表 示 します。このダイアログ ボックスからエディタを選 択 できます。
3. [**Test**] ボタンをクリックして、選 択 したエディタが構 成 されていることを確 認 してから、[**OK**] をクリックします。
4. 表 示 するエージェント ステータス情 報 を設 定 するには、[Agent Status Contents] に示 されているオプション ボックスを 1 つまたは複 数 選 択 した後 、[**OK**] をクリックします。
5. ユーザー オプションの構 成 手 順 を参 照 するには、[Help] メニューから [**Help Topics**] を選 択 し、[**How Do I...?**] を選 んでから、[**Configure user options**] を選 んでください。

# 収集パラメータの設定

scopent でデータ収集に使用される **parm** ファイルの構文を確認するときに、[Configure] メニューの [**Collection Parameters**] コマンドを使用します。**parm** ファイルの設定値を調べると、構文エラーや警告を見つけ、アプリケーションの定義に使用できる容量を確認できます。

警告やエラーが見つかり、それらを修正する場合や、**parm** ファイルのパラメータを変更または追加する場合、[Edit Parm File] 機能を使用すると **parm** ファイルを簡単に変更できます。

**parm** ファイルとそのパラメータの詳細は、「データ収集の管理」を参照してください。



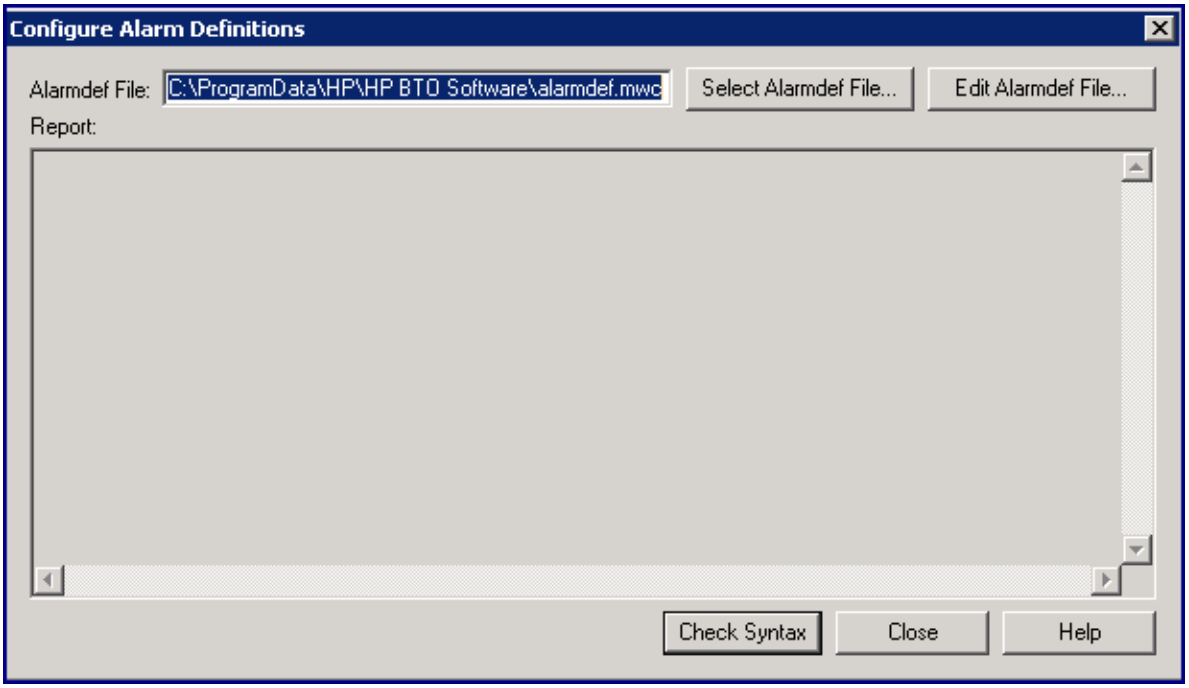

構文を確認するには、次の手順を実行します。

1. Performance Collection Component メイン ウィンドウの [Configuration] メニューから [**Collection Parameters**] をクリックします。[Configure Collection Parameters] ダイアログ ボックスが表示され、現在オープンしている **parm.mwc** ファイルの名前が [Parm File] ボックスに示されます。
2. 別の **parm** ファイルを確認するには、[**Select Parm File**] ボタンをクリックします。
3. **parm** ファイルの構文を確認するには、[**Check Syntax**] ボタンをクリックします。警告またはエラーがある場合、[Performance Collection Component] ウィンドウに表示されます。
4. **parm** ファイルのいずれかの箇所を変更する場合、[**Edit Parm File**] ボタンをクリックします。[Edit Parm File] ダイアログ ボックスと [Configure Collection Parameters] ダイアログ ボックスは、同時に使用できるように画面上に配置できます。
5. **parm** ファイルの構文確認手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?,**] を選んでから、[**Check the syntax of a collection parameters file**] を選んでください。

# アラーム定義の設定

アラーム定義ファイル (**alarmdef.mwc**) 内のアラーム定義の構文を確認するときには、[Configure] メニューから [**Alarm Definitions**] コマンドを使用します。アラーム定義の構文が正しいことを確認した後、アラーム定義と比較してログファイルを分析し、履歴ログファイルデータ内のアラームを確認できます (「ログファイルの分析」を参照してください)。

警告やエラーが見つかり、それらを修正する場合や、アラーム定義を追加または削除する場合、[Configure Alarm Definitions] ダイアログボックス内の [**Edit Alarmdef File**] ボタンを使用すると、アラーム定義ファイルを簡単に変更できます。

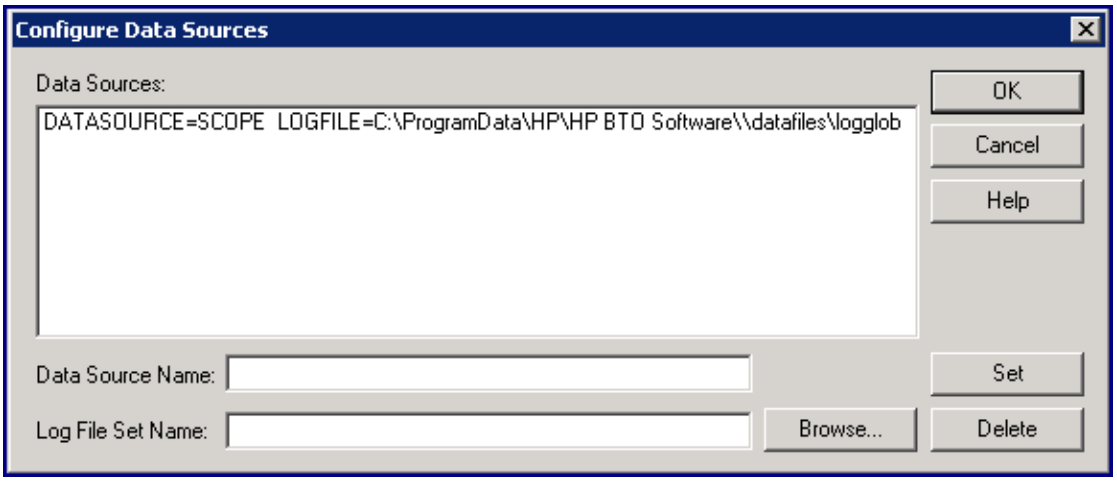



構文を確認するには、次の手順を実行します。

1. Performance Collection Component メインウィンドウの [Configure] メニューから [**Alarm Definitions**] をクリックします。[Configure Alarm Definitions] ダイアログボックスが表示され、現在オープンしているアラーム定義ファイルの名前が示されます。
2. 別のアラーム定義ファイルを確認するには、[**Select Alarmdef File**] ボタンをクリックします。
3. [**Check Syntax**] ボタンをクリックして、構文確認プロセスを開始します。数秒後、[Performance Collection Component Report Viewer] ウィンドウに警告やエラーを含めた確認結果が表示されます。
4. アラーム定義ファイルのいずれかの箇所を変更する場合、[**Edit Alarmdef File**] ボタンをクリックします。
5. アラーム定義ファイルの構文確認手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Checking the syntax of an alarm definitions file**] を選んでください。

# データ ソースの構 成

Performance Collection Component では、scopent ログ ファイルや DSI ログ ファイルなどの各 データ ソースに、固 有 のデータ ソースを使 用します。各 データ ソースは単 一 のログ ファイル セットで構 成されます。データ ソースは、**<DataDir>\conf\perf** ディレクトリにある **datasources** ファイルに構 成します。インストール後 に Performance Collection Component を最 初 に起 動 した際 には、すでに SCOPE という名 前 のデフォルト のデータ ソースが構 成されており、scopent ログ ファイル セットが使 用 できます。




## データ ソース ファイルのフォーマット

**datasources** ファイルに配 置した各 エントリは、1 つのログ ファイル セットで構 成されるデータ ソースを表 します。エントリにより示 されるデータ ソース名 は、レポジトリ サーバーを識 別 し、データの格 納 場 所 を特 定 するために使 用 されるものです。エントリは、大 文 字 と小 文 字 が区 別 されません。構 文 は次 のようになります。

**datasource=***datasource_name* **logfile=***logfile_set*

**datasource** はキーワード です。*datasource_name* は、アラーム定 義 または分 析 ソフトで使 用 するデータ ソースを識 別 するための名 前 です。データ ソース名 は固 有 のものである必 要 があります。*datasource_name* の長 さは 64 文 字 以 内 です。

**logfile** はキーワード です。*logfile_set* は、ログ ファイル セットを識 別 する完 全 修 飾 名 です。logfile_set には、scopent により作 成 された生 ログ ファイル セット、**extract** タスクにより作 成 された抽 出 ログ ファイル、DSI ログ ファイル セット のいずれかを指 定 できます。スペースを含 んだログ ファイルのパス名 を指 定 する場 合 、パス名 を二 重 引 用 符 (") で囲 む必 要 があります。

scopent ログ ファイル セットを指 定 するときには、ファイル名 **logglob** のみを使 用 してください。生 ログ ファイルは 1 つのログ ファイル セットとしてアクセスされるため、他 の生 ログ ファイル名 を指 定 する必 要 はありません。

DSI ログ ファイル セットを指 定 する場 合 も同 じです。DSI ルート ファイルの名 前 のみを指 定 してください。DSI ログ ファイル セット の他 のいずれかのファイルを指 定 する必 要 はありません。

## リモート ロケーションからのデータ ソースの構 成

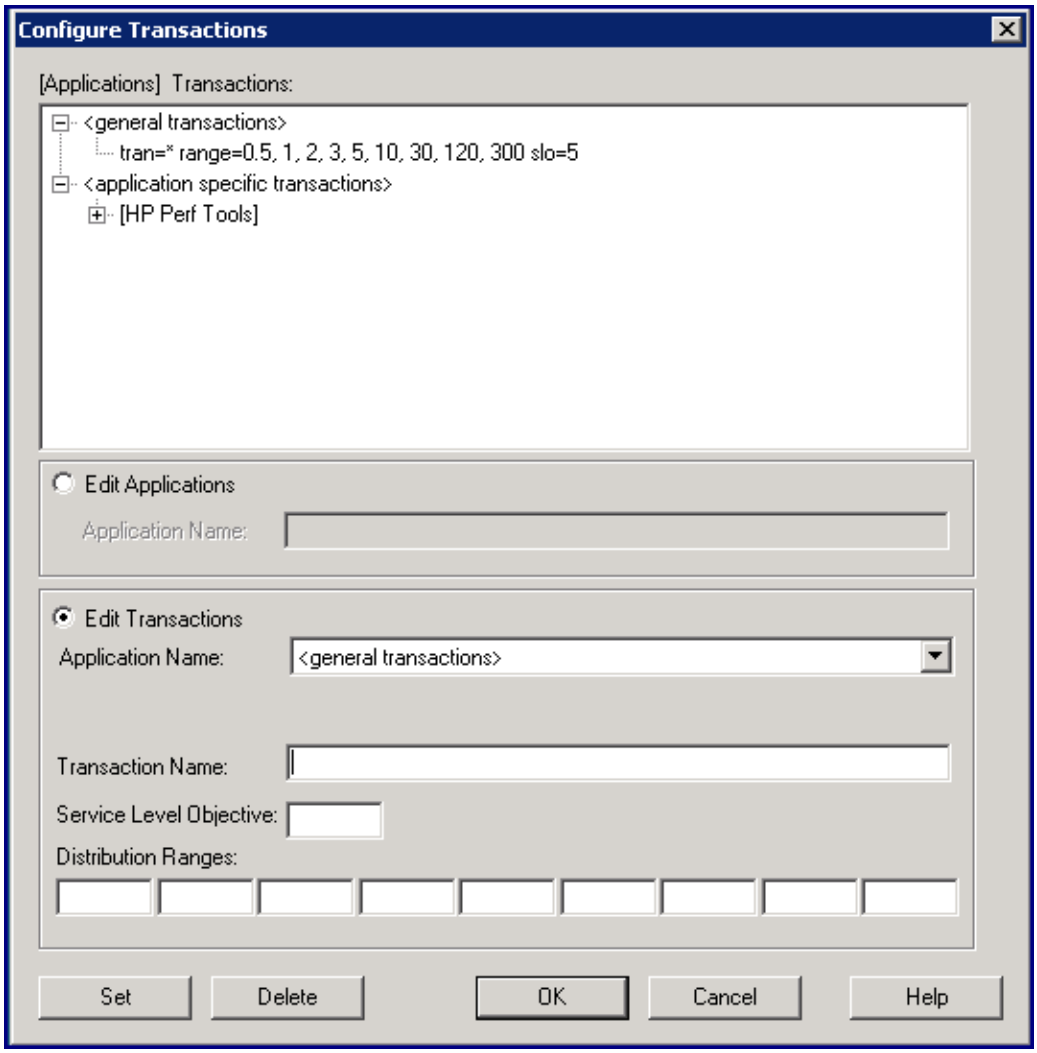
ネット ワーク化 された共 用 環 境 に存 在 するログ ファイル セットを指 定 するときには、ユニバーサル命 名 規 則 (UNC: universal naming convention) が必 要 です。Performance Collection Component サービスはシステムの起 動 時 に自 動 的 に開 始 されますが、リモート 接 続 されたファイル システムに関 するドライブ マッピングは、ユーザーがログオンするまで確 立 されません。したがって、ドライブ マッピング名 を使 用 してリモート システム上 のログ ファイルを参 照 するデータ ソースがある場 合 、Coda は無 効 なデータ ソース エラーを生 成 します。ログオンした後 にPerformance Collection Component サービスを開 始 すると、ドライブ マッピングが確 立 されているため、データ ソースは処 理 されます。

データ ソース エントリの 3 つの例 を次 に示 します。

# トランザクションの構 成

アプリケーションに合 わせてトランザクション データの収 集 をカスタマイズするときには、トランザクション構 成 ファイル **ttdconf.mwc** を使 用 します。このファイルは、トランザクション名 、パフォーマンス区 分 範 囲 、各トランザクションのサービス レベル目 標 を定 義 します。オプションで、アプリケーションに固 有 のトランザクションも定 義 できます。

デフォルト の **ttdconf.mwc** ファイルには 3 つのエントリがあります。2 つのエントリは、Performance Collection Componentscopent コレクタが使 用 するトランザクションを定 義 し、もう 1 つのエントリ tran=* は、アプリケーション応 答 測 定 (ARM: Application Response Measurement) API 関 数 コールを備 えたアプリケーション内 のすべてのトランザクションを登 録 します。




デフォルトの **ttdconf.mwc** ファイルにある tran=* エントリのサービスレベル目標と範囲値を使用するようなシステムに新しいアプリケーションを追加する場合、新しいトランザクションを組み込むために何も行う必要はありません。すべてのデフォルト値は自動的に適用されます。

ただし、固有のサービスレベル目標と区分範囲値が設定されているトランザクションを持つシステムにアプリケーションを追加する場合、それらのトランザクションを **ttdconf.mwc** ファイルに追加する必要があります。

> **注 : ttdconf.mwc** ファイル内のエントリの順序は任意です。完全に一致するエントリが最初に検索されます。エントリが見つからない場合、末尾にアスタリスク (*) があり、先頭からの一致文字数が最も多いものが使用されます。

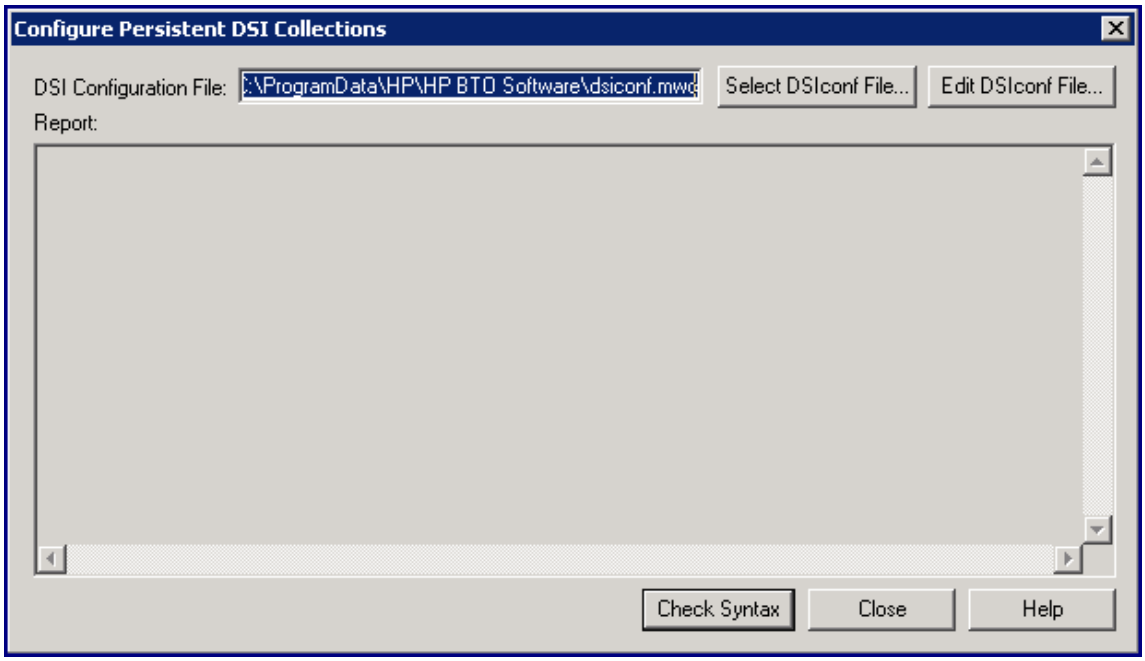
ファイルを変更する前に、「トランザクション追跡の紹介」で、構成ファイルのフォーマット、トランザクションとアプリケーションの名前、パフォーマンス区分範囲、サービスレベル目標を参考にしてください。設定するには、Performance Collection Component メインウィンドウの [Configure] メニューから [**Transactions**] をクリックすると、[Configure Transactions] ダイアログボックスが表示されます。このダイアログボックスを使用して、次の作業を実行できます。

- 一般トランザクションを追加する。
- アプリケーション固有のトランザクションを追加する。
- トランザクションのパフォーマンス区分範囲またはサービスレベル目標を変更する。
- トランザクションを削除する。

これらのタスクの実行手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Configure transactions**] を選んでください。

# Persistent DSI Collections の構成

DSI 構成ファイル **dsiconf.mwc** の構文を確認または修正するには、[Configure] メニューの [**Persistent DSI Collections**] コマンドを使用します。**dsiconf.mwc** ファイルは、外部ソースから Performance Collection Component 内に取り込まれたデータ収集の継続的な記録を構成するのに使用します。詳細については、「データソースの概要」を参照してください。




DSI 構成ファイルの構文を確認するには、以下の手順に従います。

1. Performance Collection Component メインウィンドウの [Configure] メニューから [**Persistent DSI Collections**] をクリックします。[Configure Persistent DSI Collections] ダイアログボックスが表示され、現在オープンしている **dsiconf.mwc** ファイルの名前が示されます。
2. 別の **dsiconf.mwc** ファイルを確認するには、[**Select DSIconf File**] ボタンをクリックします。
3. ファイルの構文を確認するには、[**Check Syntax**] ボタンをクリックします。警告またはエラーがある場合、[Performance Collection Component] ウィンドウに表示されます。
4. ファイルのいずれかの箇所を変更する場合、[**Edit DSIconf File**] ボタンをクリックします。[Edit DSIconf File] ダイアログボックスと [Configure Persistent DSI Collections] ダイアログボックスは、同時に使用できるように画面上に配置できます。

DSI 定義ファイルの構文確認手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I…?**] を選んでから、[**Check the syntax of a DSI configuration file**] を選んでください。

DSI 設定ファイルを変更するには、以下の手順に従います。

1. Performance Collection Component メイン ウィンドウの [Configure] メニューから [**Persistent DSI Collections**] をクリックし、[Configure Persistent DSI Collections] ダイアログボックスの [**Edit DSIconf File**] ボタンをクリックします。現在オープンしている **dsiconf.mwc** ファイルの内容が、事前に指定したエディタまたはワード プロセッサに表示されます (エディタまたはワード プロセッサを指定するには、「ユーザー オプションの構成」を参照してください)。
2. ファイルを変更する前に、「Windows での Performance Collection Component の使用」を参照して、規則や記述方法を参考にしてください。
3. 必要に応じてファイルを変更し、テキスト フォーマットで保存します。

別の作業を始める前に、**dsiconf.mwc** ファイルに対して行った変更を有効にする必要があります。次の手順を実行します。

1. [MeasureWare Services] ウィンドウを開くには、Performance Collection Component のメイン ウィンドウにある [Agent] メニューから [**Start/Stop**] をクリックします。
2. [**Persistent DSI Collections**] チェック ボックスを選択します。
3. [**Refresh**] ボタンをクリックします。
4. [**Close**] ボタンをクリックして、メイン ウィンドウに戻ります。

DSI 構成ファイルの変更手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I…?**] を選んでから、[**Modify a DSI configuration file**] を選んでください。

**注:** WordPad、Notepad、Microsoft Word のいずれかを使用して **dsiconf.mwc** ファイルを変更した後、[名前を付けて保存] コマンドを使用してファイルを保存すると、デフォルトの **.txt** 拡張子が自動的に付けられ、ファイル名は **dsiconf.mwc.txt** になります。ファイル名 **dsiconf.mwc** で保存するには、[名前を付けて保存] コマンドを使用して、ファイルをテキスト ファイルとして保存することを指定し、ファイル名を二重引用符 (") で囲みます。たとえば、"**dsiconf.mwc**" です。

# Performance Collection Component のステータス確認

Performance Collection Component プロセスの現在のステータスを調べるときには、[Agent] メニューから [**Status**] コマンドを使用します。この情報は**perfstat** プログラムにより生成されます。

ステータス レポートにどの情報を含めるかは、[Configure] メニュー画面から [**Options**] コマンドを選択し、[Configure Options] ダイアログ ボックスで次のいずれかのオプションを選択することで指定できます。

## 実行中のプロセス

Performance Collection Component に関して現在実行中のバックグラウンド プロセスおよびフォアグラウンド プロセスがリストされます。実行中でなければならないバックグラウンド プロセスは、実行中でな

くてもリストされます。

## データコム サービス

データコム サービスは、Performance Collection Component データコム サービスを特定して、そのサービスと通信します。アラーム ジェネレータ データベース サーバー (agdbserver) プロセスが実行中で応答可能な状態であるかどうかを示します。データ通信が無効になっている場合、データコム サービスの応答の待機中、この情報が生成されるまでに 30 秒以上の時間を要することがあります。

## システム サービス

Scope Collector、Transaction Manager、Measurement Interface などの Performance Collection Component システム サービスの現在のステータスを示します。

## システム構成

システム名、オペレーティング システムのバージョン、プロセッサの種類です。

## ファイル バージョン番号

Performance Collection Component ファイルのバージョン番号です。欠落している重要なファイルが通知されます。

## ステータス ファイルの最新エントリ

各パフォーマンス ツール ステータス ファイルから最新のエントリをいくつか表示します。

## ステータス ファイルの警告とエラー

パフォーマンス ツール ステータス ファイルから文字列「Error」または「Warning」を含む行をリストします。警告が長期間無視されている場合、非常に大きなリストが作成される可能性があります。

現在のステータスの一覧を表示するには、Performance Collection Component メイン ウィンドウの [Agent] メニューから [**Status**] をクリックします。[Configure Options] ダイアログ ボックスから選択した情報が、[Performance Collection Component Report Viewer] に表示されます。

すべてのステータス情報の完全なレポートを作成するには、[Agent] メニューから [**Report**] をクリックします。すべてのステータス情報の完全なリストが、[Performance Collection Component Report Viewer] に表示されます。

Performance Collection Component ステータスの確認手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I…?**] を選んでから、[**Check status of Performance Collection Component processes**] を選択します。

Windows のコマンド プロンプトからも **perfstat** プログラムを実行できます。

# パフォーマンス カウンタ収集の構築

Performance Collection Component は Windows のパフォーマンス カウンタにアクセスできます。パフォーマンス カウンタは、システム、システム上のアプリケーション、デバイス パフォーマンスを測定するのに

使用します。データ収集を構築するために特定のパフォーマンス カウンタを選択するには、拡張収集ビルダ/マネージャ (ECBM) を使用します。

# パフォーマンス カウンタ収集の構築

収集を構築するには、Performance Collection Component メイン ウィンドウの [Agent] メニューから [**Extended Collections**] を選択します。[Extended Collection Builder and Manager] ウィンドウが表示されます。左側のペインには Windows オブジェクトがリストされます。収集の構築手順を参照するには、[Extended Collection Builder and Manager] ウィンドウの [Help] メニューから [**Help Topics**] を選択してください。

Windows のパフォーマンス カウンタ収集を構築した後、最下部の [Extended Collection Manager] ペインを使用して、新規または既存の収集を登録、開始、または終了します。

# パフォーマンス カウンタ収集の管理

データ収集を管理するには、[Extended Collection Builder and Manager] の最下部にある [Extended Collection Manager] ペインを使用します。最初、収集は表示されません。データの収集を開始する前に収集を登録する必要があります。

作成した収集の登録または格納後、[Extended Collection Manager] ペインに現在の収集のリストが表示されます。[Extended Collection Manager] ペインには各収集のステータスも表示され、収集に関する情報 (プロパティ) を確認できます。収集の管理手順を参照するには、[Extended Collection Builder and Manager] ウィンドウの [**Help**] メニューから [**Help Topics**] を選択してください。

## 拡張収集ビルダ/マネージャの使用のヒント

**<Installdir>\paperdocs\mwa\C\monxref.txt** ファイルには、Windows のパフォーマンス カウンタおよびコマンドへの Performance Collection Component メトリックの相互参照が含まれています。Performance Collection Component によって収集済みのメトリックに対する拡張収集ビルダ/マネージャを介してデータを記録すると、システム オーバーヘッドが増加します。

Extended Collection Builder を使用して収集を作成する場合、Performance Collection Component の内部で使用するために、Windows のパフォーマンス カウンタにはデフォルトのメトリック名が割り当てられます。通常、これらのデフォルト名は意味を持たず、簡単に判読できません。メトリック名に意味を持たせたり、ソース アプリケーションによって提供されたメトリック名に一致させるには、[Extended Collection Builder and Manager] ウィンドウの左ペインから右ペインにメトリック名をドラッグしてから、メトリック名を右クリックまたはダブルクリックします。(詳しい手順は、拡張収集ビルダ/マネージャのオンラインヘルプを参照してください。)

61 以上の収集を起動する場合、60 を超える収集がエラー状態になります。この場合、他の収集に不具合が生じるおそれがあります。

**Wolfpack** で構成されたシステムから論理ディスク メトリックを収集する場合、収集を登録した際には存在しなかった新しいディスク インスタンスのデータを収集するために、収集を再開する必要があります。

収集を完全に削除するには、収集を削除した後に Performance Collection Component を再起動する必要があります。Performance Collection Component を再起動しない場合、削除操作中にエラーが表示される場合があります。通常このエラーは、一部のファイルが完全に削除されていないことを表します。Performance Collection Component を再起動後に、残ったファイルおよびディレクトリの手動による削除が必要となる場合があります。

キャッシュ カウンタを持つ一部のメトリックについて、拡張収集ビルダ/マネージャで欠落値が報告される場合があります。メトリック値がオーバーフローした際に、一部の環境でこの問題が発生するおそれがあります。また、メッセージが ECBM ステータス ファイルに送信されます。収集を再開するとこの問題を解決できます。

拡張収集ビルダ/マネージャの概念、データ収集の作成および表示の手順の説明は、拡張収集ビルダ/マネージャのオンライン ヘルプから参照できます。オンライン ヘルプを表示するには、デスクトップから、**スタート** ® **すべてのプログラム** ® **HP** ® **Operations エージェント**® **Performance Collection Component** ® **ECB-ECM Online Help** を選択します。[Performance Collection Component] メイン ウィンドウの [Agent] メニューから [**Extended Collections**] を選択し、[Extended Collection Builder and Manager] ウィンドウの [Help] メニューから [**Help Topics**] を選択することもできます。Extended Collection Builder and Manager に表示されるダイアログ ボックスの [**Help**] ボタンを選択すれば、オンライン ヘルプを利用できます。

## コマンド ラインからの ECBM の管理

Windows のコマンド プロンプトを使用すれば、<rpmtools>\bin ディレクトリから ECBM プログラムを実行できます。

コマンド ラインから収集を管理するには、以下のコマンドを使用します。

```
\rpmtools\bin\mwcmcmd.exe
```

各種オプションを表示するには、次のコマンドを入力します。

```
\rpmtools\bin\mwcmcmd /?
```

停止した収集を開始するには、次のコマンドを入力します。

mwcmcmd start <収集名>

可変インスタンス ポリシーから収集を開始するには、次のコマンドを入力します。

mwcmcmd start <ポリシー名> <収集名> <インスタンス> [オプション]

以下のオプションを使用できます。

-i <sampling_interval>- サンプリングの間隔を変更します (秒)

-l <logfile_path_name> - デフォルトの記録場所を変更します

-a <alarm_file> - アラーム定義ファイルを変更します

有効な収集を停止するには、次のコマンドを入力します。

mwcmcmd stop <収集名>

以下はポリシー ファイルを登録するためのコマンドです。

mwcmcmd register <ポリシー ファイル> <収集/ポリシー名> [オプション]

固定インスタンス ポリシーの登録では、以下のオプションのみが使用できます。

-i <sampling_interval> - サンプリングの間隔を変更します (秒)

-l <logfile_path_name> - デフォルトの記録場所を変更します

-a <alarm_file> - アラーム定義ファイルを変更します

単一の収集を削除する手順

mwcmcmd delete <収集/ポリシー名> [オプション]

収集の削除には、以下のオプションのみが使用できます。

`-p <archive_path> -` 特定のパスにログファイルをアーカイブします

-r - Performance エージェントを再起動します

複数の収集またはポリシーを削除する手順

mwcmcmd delete { <収集/ポリシー名> | -c | -all }

`-c -` すべての収集を削除します

`-c -` すべての収集およびポリシーを削除します

**注 :** 複数のポリシー/収集を同時に削除する場合は、Performance Collection Component が自動に再起動し、割り当てられているすべてのログファイルが削除されます。

登録済みのすべての収集をリスト表示するには、次のコマンドを入力します。

`mwcmcmd list`

収集またはポリシーのプロパティをリスト表示するには、次のコマンドを入力します。

`mwcmcmd properties <収集/ポリシー名>`

可変インスタンスオブジェクトをリスト表示するには、次のように入力します。

mwcmcmd objects <ポリシー名>

# データ ソース統 合 の概 要

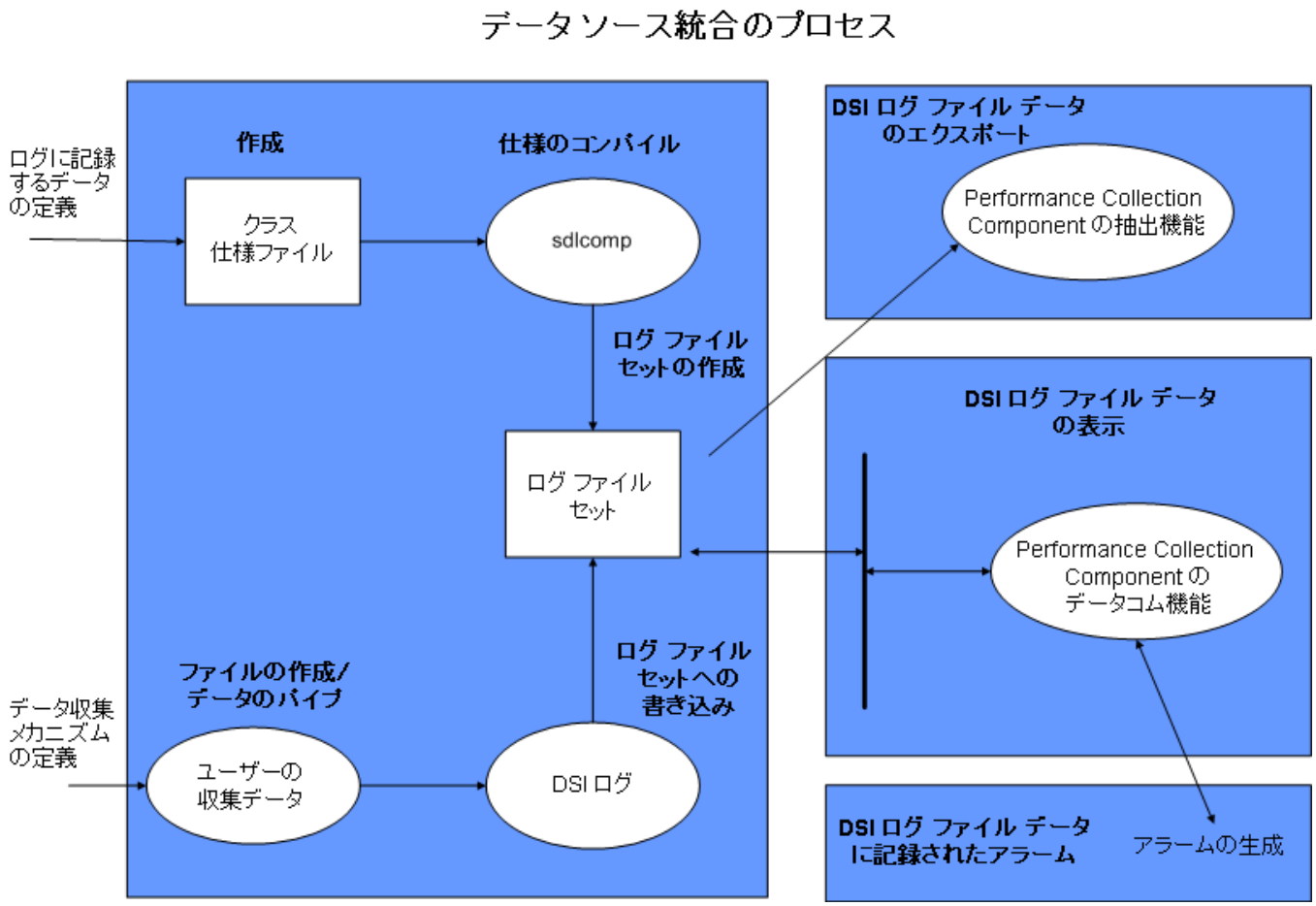
データ ソース統 合 (DSI) 技 術 により、HP Operations エージェントはデータの記 録 、アラームの定 義 、Performance Collection Component の scope コレクタが記 録 するメトリックのみでなく、新しい データ ソースのメトリックへのアクセスが可 能 になります。各 メトリックは、データベース、LAN モニタ、およびエンド ユーザーのアプリケーションなどのデータ ソースから獲 得 します。

DSI を使 用 して記 録 するデータは、scope コレクタが記 録 する標 準 のパフォーマンス メトリックと共 に HP Performance Manager で表 示 されます。また、DSI が記 録 したデータは Performance Collection Component の **extract** プログラムを使 用 して、表 計 算 ソフトや類 似 の分 析 ソフトウェアで表 示 するためにエクスポートされます。

## DSI の動 作

次 の図 は、DSI ログ ファイルを作 成 する方 法 とDSI ログ ファイルを使 用 してデータを記 録 し管 理 する方 法 を示 します。DSI ログ ファイルには、Performance Collection Component の scope コレクタ以 外 で収 集 される自 己 記 述 データが含 まれます。DSI のプロセスの詳 細 は、次 のページで説 明 します。

**図 17 データ ソース統 合 のプロセス**




DSI を使 用 してデータを記 録 するには、次 の作 業 を行 います

## クラス仕 様 の作 成

はじめに、記 録 するデータの各 クラスの仕 様 を作 成 しコンパイルします。仕 様 では、データ クラスとクラス内 で記 録 する各 メトリックについて記 述 します。DSI コンパイラ、**sdlcomp** を使 用 して仕 様 をコンパイルすると空 のログ ファイル セット が作 成 され、**dsilog** プログラムからデータを受 信 します。このプロセスにより、ルート ファイル、記 述 ファイル、および 1 つ以 上 のデータ ファイルを含 むログ ファイル セット が作 成 されます。

## データの収 集 と記 録

対 象 となるプロセスを起 動 して、記 録 するデータを収 集 します。収 集 プロセスの出 力 は、直 接 **dsilog** プログラムに渡 される場 合 と、データを保 存 するファイルを介 して渡 される場 合 があります。**dsilog** はデータを仕 様 に従 って処 理 し、適 切 なログ ファイルに書 き込 みます。**dsilog** プログラムでは、受 信 データの形 式 とフォームを指 定 できます。

DSI プロセスに送 信 するデータは、複 数 のデータ レコードを含 む必 要 があります。各 レコード は 1 行 に含 まれるメトリック値 で構 成 されます。DSI に一 度 に 1 レコード ずつデータを送 信 しプロセスを終 了 した後 、別 のレコードを送 信 する場 合 、**dsilog** は付 加 できますがデータの要 約 はできません。

## データの使 用

Performance Manager を使 用 して、DSI ログ ファイル データを表 示 できます。または、Performance Collection Component の **extract** プログラムを使 用 して、他 の分 析 ツールで使 用 するためにデータをエクスポート できます。DSI メトリックが確 定 条 件 を超 えたときにアラームを発 生 するようにも構 成 できます。

# データ ソース統合の使用方法

この章では、DSI の使用方法の概要について、次の事項を説明します。

- データ収集の計画
- クラスの仕様ファイルにおけるログ ファイル フォーマットの定義
- 空のログ ファイル セットの作成
- ログ ファイル セットへのデータの記録
- ログ データの使用

DSI クラスの仕様とDSI プログラムについての詳細は、14 章「DSI クラスの仕様」と15 章「 DSI プログラムに関する参考資料」を参照してください。

## データ収集の計画

DSI クラスの仕様ファイルを作成しログ処理を開始する前に、次の事項について確認する必要があります。

- 動作環境。コンピューティング リソースを管理する場合に有益となるデータの種類を認識するために必要です。
- 使用できるデータ
- データの保存場所
- データの収集方法
- データ項目間の区切り記号。**dsilog** が適切に処理を実行するには、入力ストリーム内のメトリック値を空白 (デフォルト)、またはユーザー定義の区切り記号で区切る必要があります。
- 収集の頻度
- ログの管理に必要な領域
- データにアクセスする場合に使用するプログラムの出力、またはプロセスの出力
- 生成するアラームと必要な条件
- クラスの仕様と**dsilog** プロセスで記録する場合に使用できるオプション

## ログ ファイル フォーマットの定義

収集するデータの種類が明確になると、クラスの仕様を作成して記録するデータを定義し、記録されたデータを含むログ ファイル セットを定義します。クラスの仕様ファイルには次の情報を入力します。

- データ クラス名とID 番号
- クラス名の代わりとなるラベル名 (オプション)。ラベル名がある場合は Performance Manager などで使用されます。
- 古いデータをロールアウトして新しいデータの領域を確保する場合の必要事項。詳細は、「ログ ファイルの編成方法」を参照してください。

- メトリック名と他の記述情報。メトリック値を考慮した 10 進数の値などが含まれます。
- 1 時間に記録されるレコード数を制限するデータ要約方法

クラスの仕様の例を次に示します。

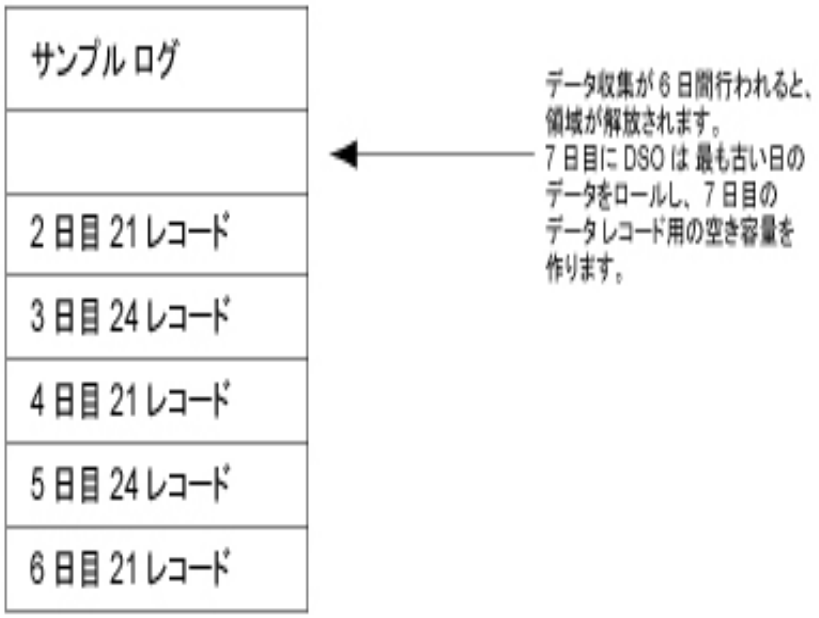
```
CLASS VMSTAT_STATS = 10001

LABEL "VMSTAT data"

INDEX BY HOUR

MAX INDEXES 12

ROLL BY HOUR

RECORDS PER HOUR 120;


METRICS

RUN_Q_PROCS = 106

LABEL "Procs in run q"

PRECISION 0;


BLOCKED_PROCS = 107

LABEL "Blocked Processes"

PRECISION 0;
```

クラスの仕様ファイルには 1 つ以上のクラスが含まれます。クラスの仕様ファイルの作成が完了すると、ファイルに名前を付けて保存します。DSI コンパイラ sdlcomp を実行する場合は、このファイルを使用してログファイルセットを作成します。クラスの仕様とメトリック記述の構文についての詳細は、14 章の「DSI クラスの仕様」を参照してください。

## ログファイルの編成方法

ログファイルはクラスに編成されます。各クラスは、受信データソースの 1 つを表現していて、共に記録されるデータ項目やメトリックのグループで構成されます。クラス内にあるデータの各レコードまたは各ラインは、メトリックグループの値のサンプルを表現します。

各クラスのデータは、ログファイルセットの一部であるログファイル内のディスクに保存されます。ログファイルセットは、ルートファイル、記述ファイル、1 つ以上のログファイルで構成されます。クラス内のデータはすべて常に単一のデータファイルに保持されます。ただし、ログファイルセット名を sdlcomp コンパイラに使用する場合は、単一のログファイルセットや個別のログファイルセットに複数のクラスを保存できます。次の図は、2 つのクラスを単一のログファイルセットに保存する方法を示します。




各 クラスは循 環 ログ ファイルとして扱 われるため、複 数 のクラスを単 一 のログ ファイル セットに保 存 するように指 定 した場 合 でも、各 クラスの記 憶 容 量 を個 別 に設 定 できます。記 憶 容 量 に達 するとクラスは「ロール」されます。「ロールする」とは、新しいデータの領 域 を確 保 するためにクラス内 で最 も古 いレコードを削 除 することです。

アーカイブ ファイルに古 いデータをエクスポートするなど、クラスがロールされたときに実 行 するアクションを指 定 できます。

# ログ ファイル セット の作 成

DSI コンパイラ sdlcomp は、クラスの仕 様 ファイルを使 用 して空 のログ ファイル セットを作 成 または更 新 します。ログ ファイル セットは、**dsilog** プログラムから記 録 済 みのデータを受 信 する場 合 に使 用 されます。

ログ ファイル セットを作 成 するには、次 の作 業 を実 行 します。

1. 適 切 な変 数 とオプションを使 用 して、`sdlcomp` を実 行 します。たとえば、次 のようになります。

   sdlcomp [-maxclass *value*] *specification_file* [*logfile_set*[*log file*]] [options]

2. 出 力 エラーを検 査 し、必 要 な場 合 は変 更 を行 います。

   sdlcomp についての詳 細 は、Compiler Syntax in Chapter 15項を参 照 してください。

# クラスの仕 様 ファイルとログ処 理 のテスト (オプション)

DSI は **sdlgendata** プログラムを使 用 して、受 信 する生 成 済 みのデータ ソースに対 するクラスの仕 様 ファイルをテストします。このプロセスの出 力 を調 べて、DSI がデータを仕 様 どおりに記 録 していることを

確認できます。sdlgendata については、15 章の「sdlgendata によるログ処理のテスト」を参照してください。

次の手順で、ログ処理用のクラスの仕様ファイルをテストします。

1. `sdlgendata` が生成したデータを **dsilog** プログラムに送信します。構文は次のようになります。

   **sdlgendata** *logfile_set class* **|** **dsilog** *logfile_set class* **-vo**

2. 出力を検査して、クラスの仕様ファイルがデータ収集プロセスのフォーマットと一致していることを確認します。**sdlgendata** プログラムの出力がユーザーのプログラムの出力と異なる場合は、出力フォーマットかクラスの指定ファイルのいずれかにエラーがあります。

3. 実際のデータを収集する前に、テスト プロセスで生成されたログ ファイルをすべて削除してください。

# ログ ファイル セットへのデータの記録

ログ ファイル セットを作成し任意でファイルをテストした後、必要な場合は Performance Collection Component 構成ファイルを更新し、**dsilog** プログラムを実行して受信データを記録します。

1. データ ソース構成ファイル **datasources** を更新し、アラームを生成するデータ ソースとして DSI ログ ファイルを追加します。

2. 特定の DSI メトリックにアラームを発生させる場合は、アラーム定義ファイル **alarmdef** を変更します。詳細については、15 章の「DSI メトリックのアラーム定義」を参照してください。

3. **-vi** オプション セットと共に **dsilog** プログラムにデータを渡して、ログ処理をテストします (このデータはクラスの仕様と一致する sdlgendata により生成されます)。このテストはオプションです。

4. データを検査して、正しく記録されたことを確認します。

5. テストが終了すると、テストに使用されたデータを削除します。

6. コマンド ラインから収集プロセスを起動します。

7. 収集プロセスからのデータを適切な変数やオプション セットと共に **dsilog** (またはその他の方法で **stdin**) に渡します。たとえば、次のようになります。

   <*program or process with variables*>| dsilog *logfile_set class*

> **注 : dsilog** プログラムは、データの連続的な流れを受け取るように考慮されています。したがって、**dsilog** が連続的な入力データを受け取るように、スクリプトを組み立てることが重要です。新しい入力データ ポイントに対して、新しい **dsilog** プロセスを作成するようなスクリプトは記述しないでください。そうした場合、複製のタイムスタンプが、**dsilog** ファイルに書かれ、ファイルを読む際に Performance Manager と perfalarm に不具合発生の原因となります。
>
> 問題のあるスクリプトや推奨されるスクリプトの例に関しては、16 章の「データ ソースの統合例」を参照してください。

**dsilog** オプションについての詳細は、15 章の「dsilog によるログ処理」を参照してください。

# ログ データの使用

DSI ログ ファイルを作成した後、Performance Collection Component の **extract** プログラムを使用してデータをエクスポートできます。DSI メトリックが確定条件を超えたときにアラームを発生するようにも

構 成 できます。

DSI ログ データを使 用 するには、次 の方 法 を実 行 します。

- 表 計 算 ソフトなどのツールのレポートに使 用 するデータをエクスポートします。
- Performance Manager などの分 析 ツールを使 用 して、エクスポートされた DSI データを表 示 します。
- HP Operations Manager または HP Network Node Manager を使 用 してアラームを監 視 します。

**メモ:** DSI ログ ファイルから抽 出 されたログ ファイルは作 成 できません。

# DSI クラスの仕様

この章では、次の参照情報についての詳細を説明します。

- クラスの仕様
- クラスの仕様構文
- クラスの仕様におけるメトリックの記述

## クラスの仕様

受信データの各ソースについて、受信データを保存するためのフォーマットを記述するクラスの仕様ファイルを作成する必要があります。ファイルを作成するには、次の項の「クラスの仕様構文」で説明するクラスの仕様言語を使用します。クラスの仕様ファイルには次の記述が含まれます。

- クラスの記述。受信データセットに名前と数値 ID を割り当て、保存するデータの量を決定し、新しいデータのための領域を確保するためにデータをロールするときを指定します。
- 各データ項目に関するメトリックの記述。メトリックの記述はデータ項目に名前を付けてそれを記述します。クラスに対して構成されている時間に複数のレコードが届く場合、データに適用する要約レベル (`RECORDS PER HOUR`) も指定します。

クラスの仕様ファイルを生成するには、ファイルを ASCII テキスト ファイルとして保存できるエディタまたはワード プロセッサを使用します。`sdlcomp` を実行してクラスの仕様ファイルをコンパイルする場合、対象となるファイル名を指定します。クラスの仕様をコンパイルすると、データの記憶領域としてログファイル セットが自動的に作成または更新されます。

クラスの仕様では、クラスに対して 1 時間あたりの保存レコード数よりも多くのレコードが届いた場合の要約方法を指定できます。たとえば、1 時間に 12 レコード (5 分間で 1 レコード) を保存するように指定したときにレコードが 1 分おきに届く場合、データ項目の一部の平均を取り、その他の項目を合計して実行件数を維持できます。

> **注 :** DSI コンパイラ `sdlcomp` はログファイル セット (名前は `logfile_set_name`) に対して次の名前のファイルを作成します。
>
> `logfile_set_name` および `logfile_set_name.desc`
>
> `sldcomp` はクラス (名前は `class_name`) に対して次のデフォルト名のファイルを作成します。
>
> `logfile_set_name.class_name`

上記の命名規則に矛盾する名前をクラスの仕様ファイルに使用しないように注意してください。`sdlcomp` が正しく機能しなくなる場合があります。

## クラスの仕様構文

`[ ]` (大かっこ) 内に示されている構文は、オプションです。`{ }` (中かっこ) 内に示されている複数の文は、それらの文のうちの 1 つを選択する必要があることを示します。斜体の語句は入力する変数名または数字を示します。クラスの仕様の最後や各メトリックの仕様の最後に付けるセミコロンの直前を除き、任意の場所でカンマを使用して構文をわかりやすく区切ることができます。文では、大文字と小文字を区別しません。

> **注:** *metric_label_name* や *class_label_name* などのユーザー定義の記述は、DSI クラス仕様の文法のキーワード要素とは必ず違うものを設定してください。

コメントは、# または // から始まります。# または // の後に続く語句は、その行の終わりまですべて無視されます。クラスの記述の後や各メトリックの記述の後にはセミコロンが必要であることに注意してください。次にクラスの仕様の各部分についての詳細と例を示します。

```
CLASS class_name = class_id_number

[LABEL "class_label_name"]

 [INDEX BY {HOUR | DAY | MONTH} MAX INDEXES number

 [[ROLL BY {HOUR | DAY | MONTH} [ACTION "action" ]

 [ CAPACITY {maximum_record_number} ]

[ RECORDS PER HOUR  number ]

;

METRICS

metric_name = metric_id_number

[LABEL "metric_label_name"]

[TOTALED | AVERAGED | SUMMARIZED BY metric_name]

[MAXIMUM metric_maximum_number]

[PRECISION { 0 | 1 | 2 | 3 | 4 | 5 }]

[TYPE TEXT LENGTH "length"]

;
```

# クラスの記述

クラスの記述を作成するには、特定のデータソースからのメトリックのグループに名前を付け、クラスの容量を指定し、容量に達したときにクラス内のデータをロールする方法を指定します。

クラスの記述は必ず CLASS キーワードで開始します。クラスの仕様の最終パラメータの後には必ずセミコロンが付きます。

# 構文

```
CLASS class_name = class_id_number
[LABEL"class_label_name"

[ INDEX BY { HOUR | DAY | MONTH } MAX INDEXES number
[ [ ROLL BY { HOUR | DAY | MONTH } [ACTION "action"]

[ CAPACITY {maximum_record_number} ]
[RECORDS PER HOUR number]

;
```

# デフォルトの設定

クラスの記述のデフォルトの設定は次のとおりです。

```
LABEL (class_name)

INDEX BY DAY

MAX INDEXES 9

RECORDS PER HOUR 12
```

デフォルトを使用するには、キーワード `CLASS` を *class_name* および数値 *class_id_number* と共に入力します。

# CLASS

クラス名とクラス ID は、特定のデータソースからのメトリックのグループを識別します。

**構文**

`CLASS` *class_name* `=` *class_id_number*

**使い方**

*class_name* と *class_ID_number* は次の要件を満たす必要があります。

- *class_name* は 20 文字以下の英数字です。名前は必ず英字で始まり、アンダースコアを使用できます (ただし、特殊文字は使用できません)。
- *class_ID_number* は 6 桁以内の数値です。
- *class_name* と *class_ID_number* はいずれも大文字と小文字の区別がありません。
- *class_name* と *class_ID_number* は定義されるクラスすべてで一意であり、Performance Collection Component の `parm` ファイル内で定義されているいずれのアプリケーションとも異なる必要があります(`parm` ファイルについての詳細は、『HP Operations エージェント for UNIX ユーザー マニュアル』の2 章を参照してください)。

**例**

```
CLASS VMSTAT_STATS = 10001;
```

# LABEL

クラスラベルはクラス全体を識別し、Performance Manager ではクラス名の代わりに使用されます。

**構文**

`[ LABEL "`*class_label_name*`" ]`

**使い方**

*class_label_name* は次の要件を満たす必要があります。

- 二重引用符で囲む必要があります。
- 48 文字以下にします。

- CAPACITY、ACTION など、DSI クラスの仕様構文のいずれのキーワード要素とも異なる必要があります。
- ラベルに二重引用符が含まれる場合、二重引用符の前に円記号 (\) を付けます。たとえば、ラベルが "\"my\" data" である場合、"my" data と入力します。
- ラベルが指定されていない場合、*class_name* がデフォルトとして使用されます。

**例**

```
CLASS VMSTAT_STATS = 10001

LABEL "VMSTAT data";
```

# INDEX BY、MAX INDEXES、ROLL BY

INDEX BY、MAX INDEXES、ROLL BY の各設定により、データを保存する方法や削除するときを指定できます。これらの設定により、保存するデータのブロック、保存するブロックの最大数、データが最大インデックス値に達した場合に廃棄するデータブロックのサイズを決定します。

**構文**

**[INDEX BY {HOUR | DAY | MONTH} MAX INDEXES** *number***]**

**[[ROLL BY {HOUR | DAY | MONTH} [ACTION "***action***"]]**

**使い方**

INDEX BY の設定により、クラスの容量に達したときにデータブロックをロールアウトできます。INDEX BY と RECORDS PER HOUR の各オプションを使用して、後述の「ログファイルサイズの制御」で説明されているように、クラスの容量を間接的に設定できます。

INDEX BY の設定は ROLL BY の設定を超えられません。たとえば、INDEX BY DAY は ROLL BY HOUR と共に使用できませんが、INDEX BY HOUR は ROLL BY DAY と共に使用できます。

ROLL BY が指定されていない場合、INDEX BY 設定が使用されます。容量に達した場合、最も古いロールインターバルで記録されたすべてのレコードが解放され、領域が再使用されます。

ACTION を指定した場合、データが廃棄 (ロール) される前に、その ACTION が実行されます。このオプションの ACTION を使用して、クラスからデータを削除する前に別の場所にエクスポートできます。データのエクスポートについての詳細は、15 章の「DSI プログラムに関する参考資料」を参照してください。

**ロールアクションに関する注記**

ACTION 文で指定された UNIX コマンドはバックグラウンドでは実行できません。ACTION 文でのコマンドの指定は長い遅延の原因となり、その間、新しいデータが記録されないため避けてください。

コマンドの長さが 2 行以上になる場合は、各ラインの始めと終わりに二重引用符を付けます。複数のラインを結合したときにコマンドラインの各オプションがつながらないように、引用符の内側には必要に応じてスペースを入れます。

コマンドに二重引用符が含まれる場合は、その直前に円記号 (\) を付けます。

ACTION は 199 文字以下に制限されます。

ACTION 文でマクロを使用して、ログファイルからロールアウトされるデータのタイムウィンドウを定義できます。これらのマクロは、dsilog により展開されます。$PT_START$ を使用してロールアウトされる

データのブロックの先頭を UNIX 時間 (1/1/70 00:00:00 からの秒数) で指定することも、$PT_END$ を使用してデータの終端を UNIX 時間で指定することも可能です。これらは、データを上書きする前にデータをエクスポートする extract プログラムと組み合わせて使用すると特に便利です。

マクロが使用される場合は、最大 199 文字まで拡張できます。

**例**

次の例は、INDEX BY、MAX INDEXES、および ROLL BY の各文節間の関係が明確になることを示します。

次の例は、CAPACITY を 144 レコード (1×12×12) に間接的に設定しています。

```
CLASS VMSTAT_STATS = 10001

LABEL "VMSTAT data"

INDEX BY HOUR

MAX INDEXES 12

RECORDS PER HOUR 12;
```

次の例は、CAPACITY を 1440 レコード (1×12×120) に間接的に設定しています。

```
CLASS VMSTAT_STATS = 10001

LABEL "VMSTAT data"

INDEX BY HOUR

MAX INDEXES 12

RECORDS PER HOUR 120;
```

次の例では、ROLL BY HOUR を示します。

```
CLASS VMSTAT_STATS = 10001

LABEL "VMSTAT data"

INDEX BY HOUR

MAX INDEXES 12

ROLL BY HOUR

RECORDS PER HOUR 120;
```

次の例は、現在 (週末を除き) ロールの対象として識別されているすべてのデータを、上書きする前に sys.sdl と呼ばれるファイルにエクスポートしています。次に示した例の最後の数行は二重引用符で囲まれており、単一のコマンドを示していることに注意してください。

```
CLASS VMSTAT_STATS = 10001

LABEL "VMSTAT data"

INDEX BY HOUR

MAX INDEXES 12

ROLL BY HOUR
```

```
ACTION "extract -xp -l sdl_new -C SYS_STATS "
"-B $PT_START$ -E $PT_END$ -f sys.sdl, purge -we 17 "
RECORDS PER HOUR 120;
```

**その他 の例**

インデックスの設 定 例を次 に示します。保 存 するデータの量 を決 定 するときに参 考 にしてください。

| INDEX BY | MAX INDEXES | 保存データ量 |
|---|---|---|
| HOUR | 72 | 3 日 分 |
| HOUR | 168 | 7 日 分 |
| HOUR | 744 | 31 日 分 |
| DAY | 365 | 1 年 分 |
| MONTH | 12 | 1 年 分 |

次 の表 では ROLL BY による設 定 の詳 細 を説 明 します。

| INDEX BY | MAX INDEXES | ROLL BY | 説 明 |
|---|---|---|---|
| DAY | 9 | DAY | 9 日 分 のデータがログ ファイルに保 存 されます。10 日 目 が記 録 される前 に、1 日 目 がロール アウトされます。これらは、インデックスおよび最 大 インデックス数 のデフォルト値 です。 |
| HOUR | 72 | HOUR | 72 時 間 (3 日 ) 分 のデータがログ ファイルに保 存 されます。73 時 間 目 が記 録 される前 に、1 時 間 目 がロール アウトされます。その後 、各 時 間 の始 まりに、「最 も古 い」時 間 がロール アウトされます。 |
| HOUR | 168 | DAY | 168 時 間 (7 日 ) 分 のデータがログ ファイルに保 存 されます。169 時 間 (8 日 ) 目 が記 録 される前 に、1 日 目 がロール アウトされます。その後 、各 日 の始 まりに、「最 も古 い」日 がロール アウトされます。 |
| HOUR | 744 | MONTH | 744 時 間 (31 日 ) 分 のデータがログ ファイルに保 存 されます。745 時 間 (32 日 ) 目 が記 録 される前 に、1 か月 目 がロール アウトされます。その後 、745 時 間 目 が記 録 される前 に、「最 も古 い」月 がロール アウトされます。<br><br>たとえば、dsilog を 4 月 15 日 に開 始 し、5 月 16 日 まで (744 時 間 ) のデータを記 録 する場 合 、745 時 間 目 (5 月 17 日 の最 初 の 1 時 間 ) のデータを記 録 する前 に、dsilog は 4 月 分 (4 月 15 日 から 30 日 分 ) のデータをロール アウトします。 |
| DAY | 30 | DAY | 30 日 分 のデータがログ ファイルに保 存 されます。31 日 目 が記 録 される前 に、1 日 目 がロール アウトされます。その後 、各 日 の始 まりに、「最 も古 い」日 がロール アウトされます。<br><br>たとえば、dsilog を 4 月 1 日 に開 始 し、4 月 中 のデータを記 録 す |

| | | | |
|---|---|---|---|
| | | | る場合、5 月 1 日 (31 日目) のデータが記録される前に 4 月 1 日のデータがロールアウトされます。 |
| DAY | 62 | MONTH | 62 日分のデータがログファイルに保存されます。63 日目が記録される前に、1 か月目がロールアウトされます。その後、63 日目が記録される前に、「最も古い」月がロールアウトされます。<br><br>たとえば、dsilog を 3 月 1 日に開始し、3 月と 4 月のデータを記録する場合、ログファイルには 61 日分のデータが保存されます。dsilog が 5 月 1 日 (62 日目) のデータを記録すると、ログファイルはいっぱいになります。dsilog は、5 月 2 日のデータを記録する前に 3 月全体をロールアウトします。 |
| MONTH | 2 | MONTH | 2 か月分のデータがログファイルに保存されます。3 か月目が記録される前に、1 か月目がロールアウトされます。その後、各月の始まりに、「最も古い」月がロールアウトされます。<br><br>たとえば、dsilog を 1 月 1 日に開始し、1 月と 2 月のデータを記録する場合、dsilog は 3 月のデータを記録する前に 1 月全体をロールアウトします。 |

# ログ ファイル サイズの制御

各クラスに保存するデータ量と新しいデータの領域を確保するために廃棄するデータ量を決定します。

クラスの容量は、INDEX BY (時間、日、月)、RECORDS PER HOUR、MAX INDEXES から計算されます。各種設定の結果を次の例に示します。

この例では、クラスの容量は 288 (24 インデックス×1 時間あたり 12 レコード) になります。

```
INDEX BY HOUR

MAX INDEXES 24

RECORDS PER HOUR 12
```

この例では、クラスの容量は 504 (7 日 ×1 日あたり 24 時間 ×1 時間あたり 3 レコード) になります。

```
INDEX BY DAY

MAX INDEXES 7

RECORDS PER HOUR 3
```

この例では、クラスの容量は 14,880 (2 か月 ×1 か月あたり 31 日 ×1 日あたり 24 時間 ×1 時間あたり 10 レコード) になります。

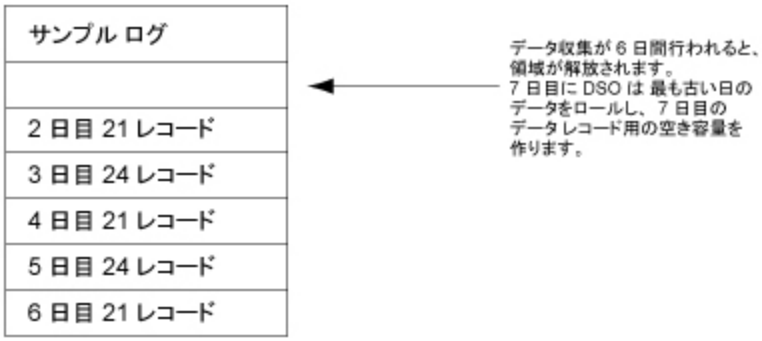
```
INDEX BY MONTH

MAX INDEXES 2

RECORDS PER HOUR 10
```

INDEX BY、RECORDS PER HOUR, および MAX INDEXES の値が指定されていない場合、DSI はクラスの記述にデフォルトを使用します。この章で前述した「クラス説明」にある「デフォルトの設定」を参照してください。

ROLL BY オプションを使用すると、クラスのレコード容量に達するたびに廃棄するデータの量を指定できます。ROLL BY に関する設定は、INDEX BY の設定に依存します。ROLL BY の単位 (時間、日、月) は必ず INDEX BY の単位よりも大きく設定されます。

次の例では、サンプル値をロールアウトする方法が示されます。

```
INDEX BY DAY

MAX INDEXES 6

ROLL BY DAY
```




上記の例では、クラスの容量が次の設定により 6 日分のデータに制限されています。

```
MAX INDEXES 6.
```

データの削除は、次の設定により 1 日分に設定されています。

```
ROLL BY DAY.
```

7 日目のデータが届くと、最も古い日のデータが廃棄されます。ログ処理の開始時には、廃棄されるデータがないことに注意してください。クラスが初めていっぱいになる 7 日目の終わりに、ロールが 1 日 1 回行われます。

# RECORDS PER HOUR

RECORDS PER HOUR の設定は、1 時間あたりのログファイルへの書き込みレコード数を決定します。RECORDS PER HOUR のデフォルト値は、Performance Collection Component のサンプルデータの測定インターバルである 5 分 (12 レコードあたり 60 分 = 5 分ごとに記録) に合わせて 12 になっています。

デフォルト値または入力する値によっては、データがログファイルの一部になる前にログ処理により要約される必要があります。各データ項目の要約に使用する方法はメトリック記述で指定します。詳細は、この章で後述している「要約方法」の項を参照してください。

**構文**

[RECORDS PER HOUR *number*]

**使い方**

ログ処理では、この値を使用して受信データを要約し、指定された数のレコードを生成します。たとえば、データが 1 分おきに到着し RECORDS PER HOUR を 6 (10 分ごと) に設定している場合、10 個のデータポイントが要約されて各レコードがクラスに書き込まれます。一般的な RECORDS PER HOUR の設定例のいくつかを次に示します。

```
RECORDS PER HOUR 6      --> 10 分あたり 1 レコード
RECORDS PER HOUR 12     --> 5 分あたり 1 レコード
RECORDS PER HOUR 60     --> 1 分あたり 1 レコード
RECORDS PER HOUR 120    --> 30 秒あたり 1 レコード
```

**注記**

RECORDS PER HOUR は、dsilog の -s seconds オプションにより無効にできます。ただし、元の設定を無効にすると Performance Manager によるデータのグラフ作成で問題が発生する場合があります。

dsilog が全記録期間中にメトリックデータを受信しない場合、そのメトリックのデータがないことを示すインジケータが記録されます。dsilog の -asyn オプションを使用して、DSI に対して最後に記録された値を使用するように指定できます。-asyn オプションの詳細は、15 章の「dsilog によるログ処理」を参照してください。

**例**

この例では、レコードは 10 分おきに書き込まれます。

```
CLASS VMSTAT_STATS = 10001
LABEL  "VMSTAT data"
RECORDS PER HOUR  6;
```

# CAPACITY

CAPACITY は、クラスに保存されるレコード数です。

**構文**

[CAPACITY {*maximum_record_number*}]

**使い方**

クラスの容量は、RECORDS PER HOUR、INDEX BY および MAX INDEXES 内の設定から引き出されます。これらの他の設定から引き出される値より大きな容量を指定する場合を除き、CAPACITY の設定は無視されます。大きな容量を指定する場合、指定する容量を確保するために MAX INDEXES の設定が増やされます。

**例**

```
INDEX BY DAY
MAX INDEXES 9
RECORDS PER HOUR 12
CAPACITY 3000
```

上記の例では、クラスの容量として 2,592 レコード (9 日 ×1 日あたり 24 時間 ×1 時間あたり 12 レコード) が引き出されます。

3000 は 2592 よりも大きいため、sdlcomp は MAX INDEXES を 11 に増やします。その結果、クラス容量は 3168 になります。コンパイル後は、-decomp オプションと共に sdlutil を実行することで、最終的な MAX INDEXES および CAPACITY の値を確認できます。

# メトリックの記述

クラスの仕様ファイルのメトリックの記述を使用して、クラスの個々のデータ項目を定義します。メトリックの記述によりメトリック名と数値の ID が対応付けられ、1 時間あたりのレコード数が RECORDS PER HOUR の設定で指定された値を超えた場合のデータ要約方法が指定されます。

> **注:** *metric_label_name* などのユーザー定義の記述は、DSI クラス仕様の文法のキーワード要素とは必ず違うものを設定してください。

dsilog フォーマット ファイルでは、メトリックの数は 100 以下に制限されることに注意してください。

```
METRICS

metric_name = metric_id_number

[LABEL "metric_label_name"]

[TOTALED | AVERAGED | SUMMARIZED BY metric_name]

[MAXIMUM metric_maximum_number]

[PRECISION { 0 | 1 | 2 | 3 | 4 | 5 }]

TYPE TEXT LENGTH "length"
```

> **注:** 数値メトリックの場合、要約方法 (TOTALED、AVERAGED、SUMMARIZED BY)、および PRECISION を指定できます。テキスト メトリックの場合、TYPE TEXT LENGTH のみを指定できます。

# METRICS

メトリック名と ID 番号は、収集されるメトリックを識別します。

**構文**

METRICS
*metric_name* = *metric_id_number*

**使い方**

メトリック セクションは、最初のメトリック定義の前に METRICS キーワードで開始する必要があります。各メトリックには、次の要件を満たす名前が必要です。

- 20 文字以下にします。
- 英字で始めます。

- 英 数 字 とアンダースコアでのみ構 成 します。
- 大 文 字 と小 文 字 の区 別 はありません。

また、メトリックには 6 桁 以 下 のメトリック ID があります。

*metric-_name* および *metric_id_number* はそれぞれクラス内 で定 義 したすべてのメトリックの間 で一 意 である必 要 があります。*class_name*:*metric_name* の組 み合 わせはこのシステムにおいて一 意 であり、いずれの *application_name*:*metric_name* とも異 なる必 要 があります。

各 メトリックの記 述 の終 わりにはセミコロン (;) を付 けます。

定 義 が同 一 である場 合 、同 じログ ファイル セット 内 でデータが保 存 されている別 のクラスのメトリック 名 を再 使 用 できます (13 章「ログ ファイル フォーマット の定 義 」を参 照 してください)。同 じログ ファイル セット 内 の別 のクラスですでに定 義 されているメトリックの定 義 を再 使 用 するには *metric_name* のみを指 定 し、*metric_id_number* やその他 の仕 様 を指 定 する必 要 はありません。いずれかのオプションを前 回 定 義 したメトリックと異 なるように設 定 する場 合 、そのメトリックに一 意 の名 前 と数 値 ID を割 り当 て、再 定 義 する必 要 があります。

クラスの仕 様 のセクションにおけるメトリック名 の順 序 は、ログ データをエクスポートするときのフィールド の順 序 を決 定 します。受 信 データの順 序 がこの仕 様 でリストされる順 序 と異 なる場 合 や、受 信 デー タ ストリーム内 のすべてのデータを記 録 する必 要 がない場 合 、15 章「DSI プログラムに関 する参 考 資 料 」に記 載 されている適 正 な位 置 にメトリックをマップする方 法 を参 照 してください。

タイムスタンプ メトリックは、各 クラスの最 初 のメトリックとして自 動 的 に挿 入 されます。エクスポートされたデータ内 でタイムスタンプを別 の位 置 に表 示 する場 合 、内 部 的 に確 定 しているメトリック定 義 の短 い形 式 (`DATE_TIME`;) をタイムスタンプの表 示 位 置 に含 めます。タイムスタンプを除 き、受 信 データの 一 部 である UNIX タイムスタンプ (1/1/70 00:00:00 からの秒 数 ) を使 用 するには、`dsilog` プロセスを開 始 するときに `-timestamp` オプションを選 択 します。

メトリック名 をラベルとして使 用 し、`AVERAGED`、`MAXIMUM 100`、および `PRECISION` 3 の小 数 点 位 置 のデフォルトを使 用 する最 も単 純 なメトリックの記 述 では、次 の記 述 が必 要 です。

```
METRICS

metric_name = metric_id_number
```

> **注 :** メトリック名 を再 使 用 しているかどうかにかかわらず、`sdlcomp` を使 用 して各 クラスをコンパイ ルし `dsilog` プロセスを使 用 して、そのクラスのデータの記 録 を開 始 する必 要 があります。

**例**

```
VM;
```

`VM` は、同 じログ ファイル セット 内 の別 のクラスですでに定 義 されているメトリックの定 義 を再 使 用 して いるメトリックの例 です。

# LABEL

メトリック ラベルは、Performance Manager グラフとエクスポート データのメトリックを識 別 します。

**構 文**

[LABEL "*metric_label_name*"]

**使 い方**

グラフとエクスポート データのメトリックのラベルとなるテキスト文字列を、二重引用符で囲んで指定します。48 文字まで入力できます。ラベルを指定しない場合、メトリック名がメトリックを識別するために使用されます。

**注記**

ラベルに二重引用符が含まれる場合は、その直前に円記号 (\) を付けます。たとえば、ラベルが `"\"my\" data"` である場合、`"my" data` と入力します。

*metric_label_name* は、`CAPACITY` や `ACTION` など、DSI クラスの仕様構文のいずれのキーワード要素とも異なる必要があります。

**例**

```
METRICS

RUN_Q_PROCS = 106

LABEL "Procs in run q";
```

# 要約方法

要約方法では、`CLASS` セクションの `RECORDS PER HOUR` オプションで設定されたレコード数を超えたレコードが届いた場合に、データを要約する方法を決定します。たとえば、実際のメトリックを合計するときに割合を平均する場合があります。要約方法は数値メトリックに対してのみ有効です。

**構文**

[{TOTALED | AVERAGED | SUMMARIZED BY *metric_name*}]

**使い方**

`SUMMARIZED BY` は、メトリックを時間ではなくクラス内の別のメトリックで平均する場合に使用します。たとえば、`TOTAL_ORDERS` と `LINES_PER_ORDER` の各メトリックを定義していると仮定します。これらのメトリックは 5 分ごとにログ処理されますが、レコードが 1 時間に 1 回しか書き込まれないとしても、`LINES_PER_ORDER` は全ライン数を全オーダー数で割った値になるように正確に要約するため、ログ処理では次の計算を 5 分ごとに行う必要があります。

- 5 分のインターバルが終わるたびに、`LINES_PER_ORDER` と `TOTAL_ORDERS` をかけて、その結果を全ラインの内部実行回数として保持します。
- `TOTAL_ORDERS` の実行回数を保持します。
- 1 時間後に全ライン数を `TOTAL_ORDERS` で割ります。

この種の計算を指定するには、`LINES_PER_ORDER` を `SUMMARIZED BY TOTAL_ORDERS` として指定します。

要約方法を指定しない場合、メトリックはデフォルトで `AVERAGED` になります。

**例**

```
METRICS

ITEM_1_3 = 11203

LABEL "TOTAL_ORDERS"

TOTALED;
```

```
ITEM_1_5 = 11205

LABEL "LINES_PER_ORDER"

SUMMARIZED BY ITEM_1_3;
```

# PRECISION

PRECISION はメトリック値の小数点以下の桁数を識別します。PRECISION を指定しない場合は、指定した MAXIMUM に基づいて計算します。いずれの値も指定しない場合、PRECISION はデフォルトの 3 になります。この設定は数値メトリックに対してのみ有効です。

**構文**

```
[PRECISION{0|1|2|3|4|5}]
```

**使い方**

PRECISION の設定により記録可能な最大値が決まります。整数には PRECISION 0 を使用します。

| PRECISION | 小数点以下の桁数 | 最大許容値 | MAXIMUM |
|---|---|---|---|
| 0 | 0 | 2,147,483,647 | > 10,000 |
| 1 | 1 | 214,748,364.7 | 1001 ～ 10,000 |
| 2 | 2 | 21,474,836.47 | 101 ～ 1,000 |
| 3 | 3 | 2,147,483.647 | 11 ～ 1,000 |
| 4 | 4 | 214,748.3647 | 2 ～ 10 |
| 5 | 5 | 21,474.83647 | 1 |

**例**

```
METRICS

RUN_Q_PROCS = 106

LABEL "Procs in run q"

PRECISION 1;
```

# TYPE TEXT LENGTH

TYPE TEXT LENGTH の 3 つのキーワードは、メトリックが数値ではなくテキストであることを指定します。テキストは、^d、\n または区切り記号 (使用している場合) 以外の任意の文字として定義します。

dsilog に入力するデータ項目間のデフォルトの区切り記号はスペースであるため、テキストにスペースが含まれる場合、区切り記号を変更する必要があります。異なる区切り記号を指定するには、15 章「DSI プログラムに関する参考資料」で説明するように dsilog -c char オプションを使用します。

**構文**

[TYPE TEXT LENGTH *length*]

**使い方**

*length* はゼロより大きく、4096 より小さい数である必要があります。

**注記**

要約方法、MAXIMUM、および PRECISION はいずれもテキスト メトリックでは指定できません。テキストは要約できないため、dsilog はインターバル内で最初に記録された値を取り込み、残りを無視します。

**例**

```
METRICS
text_1 = 16
LABEL "first text metric"
TYPE TEXT LENGTH 20
;
```

# クラス仕様のサンプル

```
CLASS VMSTAT_STATS = 10001
LABEL  "VMSTAT data"
INDEX BY HOUR
MAX INDEXES      12
ROLL BY HOUR
RECORDS PER HOUR  120;

METRICS

RUN_Q_PROCS       = 106
LABEL     "Procs in run q"
PRECISION 0;

BLOCKED_PROCS     = 107
LABEL     "Blocked Processes"
PRECISION 0;

SWAPPED_PROCS     = 108
LABEL     "Swapped Processes"
```

```
PRECISION 0;

AVG_VIRT_PAGES    = 201
LABEL     "Avg Virt Mem Pages"
PRECISION 0;

FREE_LIST_SIZE    = 202
LABEL     "Mem Free List Size"
PRECISION 0;

PAGE_RECLAIMS     = 303
LABEL     "Page Reclaims"
PRECISION 0;

ADDR_TRANS_FAULTS = 304
LABEL     "Addr Trans Faults"
PRECISION 0;

PAGES_PAGED_IN    = 305
LABEL     "Pages Paged In"
PRECISION 0;

PAGES_PAGED_OUT   = 306
LABEL     "Pages Paged Out"
PRECISION 0;
PAGES_FREED       = 307
LABEL     "Pages Freed/Sec"
PRECISION 0;

MEM_SHORTFALL     = 308
LABEL     "Exp Mem Shortfall"
PRECISION 0;
```

```
CLOCKED_PAGES     = 309
LABEL     "Pages Scanned/Sec"
PRECISION 0;

DEVICE_INTERRUPTS = 401
LABEL     "Device Interrupts"
PRECISION 0;

SYSTEM_CALLS      = 402
LABEL     "System Calls"
PRECISION 0;

CONTEXT_SWITCHES  = 403
LABEL     "Context Switches/Sec"
PRECISION 0;

USER_CPU          = 501
LABEL     "User CPU"
PRECISION 0;

SYSTEM_CPU        = 502
LABEL     "System CPU"
PRECISION 0;

IDLE_CPU          = 503
LABEL     "Idle CPU"
PRECISION 0;
```

# DSI プログラムに関する参考資料

この章では、次の参照情報についての詳細を説明します。

- sdlcompコンパイラ
- datasources 構成ファイルとalarmdef 構成ファイル
- dsilog によるログ処理
- Performance Collection Component の extract プログラムによる DSI データのエクスポート
- データソース管理ユーティリティ sdlutil

## sdlcomp コンパイラ

sdlcomp コンパイラは、クラスの仕様ファイルのエラーを検査します。エラーが検出されない場合は、ユーザーが指定するログファイルセットの記述ファイルにクラスおよびメトリックの記述が追加されます。また、sdlcomp コンパイラは、データの保存に使用するログファイルに対してログファイルセットのルートファイルのポインタをセットアップします。ログファイルセットまたはログファイルのいずれかがない場合は、コンパイラにより作成されます。

**注 :** コンパイラコマンド内でフルパスを指定すると、DSI ファイルをシステム上の任意の場所に配置できます。ただし、パスを指定した後に、DSI ログファイルを異なるディレクトリに*移動しないようにしてください*。SDL62 は、17 章「SDL エラー メッセージ」に記載されている関連クラス仕様エラー メッセージです。クラス仕様エラー メッセージ用に DSI で使用するフォーマットは、接頭辞 SDL (Self Describing Logfile)+「メッセージ番号」です。

### コンパイラ構文

sdlcomp [-maxclass *value*] specification_file

[logfile_set[log file]] [*options*]

| 変数とオプション | 定義 |
|---|---|
| -maxclass *value* | 新しいログファイルセットを作成するときに入力されるクラスの最大数を指定できます。このオプションは既存のログファイルセット名と同時に使用すると無視されます。各追加クラスは、そのクラスが使用されているかどうかにかかわらず、オーバーヘッド内で約 500 バイトのディスク領域を消費します。-maxclass を指定していない場合デフォルトは 10 です。 |
| specification_file | クラスの仕様を含むファイル名です。このファイルが現在のディレクトリにない場合、完全修飾名を指定する必要があります。 |
| logfile_set | 対象となるクラスを追加するログファイルセットの名前です。 |
| log file | 対象となるクラスに関するデータを含むセット内のログファイルです。ログファイルを指定しない場合、新しいログファイルがクラスに対して作成され、自動的に名前が付けられます。 |

| | |
|---|---|
| -verbose | コンパイラ出力の詳細説明を stdout に出力します。 |
| -vers | バージョン情報を表示します。 |
| -? | 構文の説明を表示します。 |
| -u | 1 秒あたりに複数のレコードを記録できます。このオプションは、要約されていないデータのみを記録するのに使用します。 |

## コンパイラ出力のサンプル

次のコマンド ラインを仮定します。

```
->sdlcomp vmstat.spec sdl_new
```

次のコードは正常終了したコンパイルの出力サンプルです。vmstat.spec は3 章で示した仕様ファイルのサンプルであることに注意してください。

```
sdlcomp

Check class specification syntax.


CLASS VMSTAT_STATS = 10001

LABEL  "VMSTAT data"

INDEX BY HOUR

MAX INDEXES      12

ROLL BY HOUR

RECORDS PER HOUR  120;


METRICS


RUN_Q_PROCS       = 106

LABEL     "Procs in run q"

PRECISION 0;


BLOCKED_PROCS     = 107

LABEL     "Blocked Processes"

PRECISION 0;


SWAPPED_PROCS     = 108

LABEL     "Swapped Processes"
```

```
PRECISION 0;

AVG_VIRT_PAGES    = 201
LABEL     "Avg Virt Mem Pages"
PRECISION 0;

FREE_LIST_SIZE    = 202
LABEL     "Mem Free List Size"
PRECISION 0;

PAGE_RECLAIMS     = 303
LABEL     "Page Reclaims"
PRECISION 0;

ADDR_TRANS_FAULTS = 304
LABEL     "Addr Trans Faults"
PRECISION 0;

PAGES_PAGED_IN    = 305
LABEL     "Pages Paged In"
PRECISION 0;

PAGES_PAGED_OUT   = 306
LABEL     "Pages Paged Out"
PRECISION 0;

PAGES_FREED       = 307
LABEL     "Pages Freed/Sec"
PRECISION 0;

MEM_SHORTFALL     = 308
LABEL     "Exp Mem Shortfall"
PRECISION 0;
```

```
CLOCKED_PAGES     = 309
LABEL     "Pages Scanned/Sec"
PRECISION 0;

DEVICE_INTERRUPTS = 401
LABEL     "Device Interrupts"
PRECISION 0;

SYSTEM_CALLS      = 402
LABEL     "System Calls"
PRECISION 0;

CONTEXT_SWITCHES  = 403
LABEL     "Context Switches/Sec"
PRECISION 0;

USER_CPU          = 501
LABEL     "User CPU"
PRECISION 0;

SYSTEM_CPU        = 502
LABEL     "System CPU"
PRECISION 0;

IDLE_CPU          = 503
LABEL     "Idle CPU"
PRECISION 0;
Note:  Time stamp inserted as first metric by default.

Syntax check successful.

Update SDL sdl_new.
```

```
Open SDL sdl_new

Add class VMSTAT_STATS.

Check class VMSTAT_STATS.
```

Class VMSTAT_STATS successfully added to log file set.

エラー メッセージと回 復 に関 する説 明 は、17 章 の「エラー メッセージ」を参 照 してください。

# 構 成ファイル

データの記 録 を開 始 する前 に、2 つの Performance Collection Component 構 成 ファイルを更 新 する必 要 があります。

- `/var/opt/OV/conf/perf/datasources`
- `/var/opt/perf/alarmdef` — `alarmdef` 構 成 ファイルの使 用 に関 する詳 細 情 報 は、次 の項 を参 照 してください。

# DSI メトリックのアラーム定 義

Performance Collection Component を使 用 すると、DSI メトリックに関 するアラームを定 義 できます。アラームは、DSI メトリックが定 義 条 件 を満 たす場 合 や超 える場 合 にユーザーに通 知 します。アラームを定 義 するには条 件 を指 定 し、その条 件 を満 たす場 合 や超 える場 合 に、アラートの通 知 またはアクションをトリガします。Performance Collection Component システム上 の `alarmdef` ファイルにある別 の Performance Collection Component メトリックのアラームと同 様 に、DSI により記 録 されたデータのアラームを定 義 します。`alarmdef` ファイルは Performance Collection Component の `var/opt/perf/` 構 成 ディレクトリに配 置 されています。

アラーム定 義 内 の DSI メトリック名 は必 ず完 全 修 飾 名 で指 定 します。次 に示 すように、DSI メトリック名 の直 前 に *datasource_name* と *class_name* を指 定 します。

*datasource_name* : *class_name* : `metric_name`

- *datasource_name* は、`datasources` ファイル内 でデータ ソースを構 成 するために使 用 している名 前 です。
- *class_name* は、データ ソースのクラスの仕 様 でクラスを識 別 するために使 用 している名 前 です。クラスの仕 様 でメトリック名 が一 意 である (再 使 用 されていない) 場 合 、*class_name* を入 力 する必 要 はありません。
- `metric_name` は、データ ソースのクラスの仕 様 に含 まれるデータ項 目 です。

ただし、メトリックに完 全 修 飾 名 を指 定 しない場 合 は、使 用 するデータ ソースを識 別 するために `alarmdef` ファイルに USE 文 を含 む必 要 があります。USE 文 に関 する詳 細 は、『HP Operations エージェント for UNIX ユーザー マニュアル』7 章 「パフォーマンス アラーム」を参 照 してください。

アラーム ジェネレータが読 み込 めるように `alarmdef` ファイルの変 更 をアクティブにするには、コマンド ラインに `ovpa restart alarm` コマンドを入 力 します。

アラーム定 義 構 文 、アラームの処 理 方 法 、およびアラーム定 義 のカスタマイズに関 する詳 細 は、『HP Operations エージェント for UNIX ユーザー マニュアル』7 章 を参 照 してください。

# アラーム処理

dsilog はデータを記録すると、そのデータを alarmdef ファイル内のアラート定義と比較して条件を満たしていること、または超えていることを確認します。データが条件を満たしている場合または超えている場合、アラートの通知またはアクションがトリガされます。

アラームの通知を送信する場所や、ローカルアクションの実行の有無が構成できます。アラームの通知を中央 Performance Manager 分析システムに送信し、そこでシステムパフォーマンスの特徴を示すメトリックのグラフを作成できます。SNMP トラップは、HP Network Node Manager に送信できます。Performance Collection Component システム上で、ローカルな動作を実行できます。アラーム情報も Operations Manager に送信できます。

# dsilog によるログ処理

dsilog 処理では、データにアクセスするためにユーザーが独自のプログラムを作成するか、既存のプログラムを使用する必要があります。データを dsilog に渡すとデータをログファイルセットに記録できます。定義する各クラスに対して個別のログ処理を実行する必要があります。

dsilog プログラムは stdin からデータを受信することを予測します。ログ処理を開始するには、次の例で示すように、データを収集する処理の出力を dsilog に渡します。

```
vmstat 60 | dsilog logfile_set class
```

コマンドラインで使用できるのは 1 つのパイプ (|) のみです。これは、パイプを 2 つ使用すると、UNIX バッファリングにより最初のコマンドの出力が遅延するためです。2 つ目のコマンドが実行されログファイルにデータが渡されるまで 8000 文字が書き込まれます。

fifo (指定済みのパイプ) を使用することもできます。たとえば、次のようになります。

```
mkfifo -m 777 myfifo

dsilog logfile_set class -i myfifo &

vmstat 60 > myfifo &
```

& により、バックグラウンドで処理が実行されます。

dsilog 処理を多数実行する場合、UNIX カーネルパラメータ shmmni と nflocks の値を増やす必要があることに注意してください。shmmni は共有メモリの最大セグメント数を指定し、nflocks はシステム上の最大ファイルロック数を指定します。どちらのパラメータもデフォルト値は 200 です。アクティブな DSI ログファイルセットは、それぞれ 1 つの共有メモリのセグメント (shmmni) と 1 つ以上のファイルロック (nflocks) を使用します。HP-UX では、システム管理および保守ユーティリティ (SAM) を使用して、shmmni と nflocks の設定を変更できます。

# 構文

dsilog logfile_set class [*options*]

dsilog パラメータとオプションについては、次のページ以降で説明します。

**表 1:dsilog パラメータおよびオプション**

| 変数とオプション | 定義 |
|---|---|

| | |
|---|---|
| logfile_set | データを保存するログファイルセットの名前です。現在のディレクトリにない場合、完全修飾名である必要があります。 |
| class | 記録されるクラスの名前です。 |
| -asyn | データが RECORDS PER HOUR の比率で非同期的に届くように指定します。ログインターバル中にデータが届かない場合、最後のログインターバルのデータが繰り返されます。ただし、dsilog がまだデータを記録していない場合、記録されたメトリック値は欠落データとして扱われます。これにより直線がデータのグラフィック表示に描画され、データがエクスポートされるときに各レコードのデータが繰り返されます。 |
| -c char | 指定文字を文字列の区切り記号として使用します。整数、マイナス符号、^z、\n は区切り記号として使用できません。いずれかのテキスト メトリックにスペースが含まれている場合、このオプションを指定して、固有の区切り記号を指定する必要があります。 |
| -f *format file* | ログ処理に入力されるデータを説明するファイルを指定します。このオプションを指定しない場合、dsilog は次の仮定を用いてクラスの仕様から入力のフォーマットを派生させます。詳細は、この章で後述している「フォーマット ファイルの作成」を参照してください。<br><br>入力レコード内の各データ項目は、クラスの仕様で定義されているメトリックに対応しています。<br><br>メトリックは、入力レコード内でデータ項目として表示される順序で、クラスの仕様に定義されています。<br><br>メトリックの定義より多くのデータ項目が入力レコードにある場合、dsilog は追加のデータ項目をすべて無視します。 |
| -f *format file* (続き) | 入力データ項目より多くのメトリックの定義をクラスの仕様でリストしている場合、このフィールドは、データをエクスポートするときに「欠落」データを表示し、分析ソフトウェアでデータをグラフ化するときには、そのメトリックにデータを使用できません。<br><br>フォーマット ファイル内のフィールドの数は、100 までに制限されています。 |
| -ififo または ASCII ファイル | 指定の fifo または ASCII ファイルから入力されることを示します。このオプションを使用しない場合、stdin から入力されます。このオプションを使用する場合は、収集処理を開始する前に dsilog を起動します。fifo の使用についての詳細は、man ページの mkfifo を参照してください。また、使用例については、16 章の「データ ソースの統合例」を参照してください。 |
| -s seconds | データを要約するまでの秒数です。-s オプションは、クラス スペック中の RECORDS PER HOUR への要約間隔および要約レートのデフォルトを変更します。また、このオプションは RECORDS PER HOUR の値を変更します。<br><br>ゼロ (0) は要約を終了し、すべての受信データが記録されます。-s 0 オプションは注意が必要です。dsilog は、そのポイントになったとき、ログデータにタイムスタンプします。これは、通常の間隔でタイムスタンプを行う Performance Manager と perfalarm の不具合になります。ログファイルが Performance Manager によってアクセスされる場合、-s 0 オプションは使用しないことをお勧めします。 |

| | |
|---|---|
| -t | stdout に記録されるすべてのデータを ASCII フォーマットで印刷します。 |
| -timestamp | ログ処理がタイムスタンプを提供せず、入力データですでに提供されているタイムスタンプを使用することを示します。受信データ内のタイムスタンプは、UNIX タイムスタンプフォーマット (1/1/70 00:00:00 からの秒数) で、各地の時間を表す必要があります。 |
| -vi | dsilog により入力をフィルタ処理し、エラーをログファイルではなく stdout に書き込みます。これは、記録された実データを stdout に書き込みません (次の -vo オプションを参照)。これを使用すると、入力の妥当性を確認できます。 |
| -vo | dsilog により入力をフィルタ処理し、記録された実データとエラーをログファイルではなく stdout に書き込みます。これを使用すると、データ要約の妥当性を確認できます。 |
| -vers | バージョン情報を表示します。 |
| -? | 構文の説明を表示します。 |

# dsilog によるデータ処理方法

dsilog プログラムは各入力データ文字列を走査し、区切られたフィールドを個別の数値メトリックまたはテキスト メトリックに分割します。データの処理方法を予測するための重要な規則は、入力文字列の妥当性です。有効な入力文字列では、区切り記号がすべての指定メトリックタイプ (数値またはテキスト) 間に含まれる必要があります。デフォルトの区切り記号はスペースですが、dsilog -c char コマンド ライン オプションを使用すると別の区切り記号を指定できます。

DSI がレコードを正しく解釈するために、DSI に送り込まれるすべてのレコードの最後に新しい文字列を含む必要があります。

# sdlgendata によるログ処理のテスト

データの記録を開始する前に sdlgendata プログラムを使用して、コンパイルされたログファイル セットとログ処理をテストできます。sdlgendata はクラスのメトリックを検出し、クラス内の各メトリックについてデータを作成します。

**構文**

sdlgendata logfile_set class [*options*]

sdlgendata のパラメータとオプションを次に示します。

**表 2:**Sdlgendata **パラメータおよびオプション**

| 変数とオプション | 定義 |
|---|---|
| logfile_set | データを生成するログファイル セットの名前です。 |

| | |
|---|---|
| class | データを生成するデータクラスです。 |
| -timestamp [*number*] | データと共にタイムスタンプを示します。負の数字を入力する場合、または数字を入力しない場合、現在の時刻が timestamp に使用されます。正の数字を使用すると時間は 0 から始まり、個別の新規データレコードについて *number* で示す数だけ増加します。 |
| -wait *number* | 生成されるレコード間で、*number* で示す秒数分待機します。 |
| -cycle number | *number* で示すサイクル回数の後に、データを再利用します。 |
| -vers | バージョン情報を表示します。 |
| -? | 構文の説明を表示します。 |

-vi または -vo オプションを使用して sdlgendata の出力を dsilog プロセスに渡すことにより、独自のプロセスまたはプログラムによる記録を開始する前に入力 (-vi) や出力 (-vo) を検査できます。

**注:** テストが終了すると、テストで作成したログファイルはすべて削除します。削除しない場合、これらのファイルはログファイルセットの中に残ります。

次のコマンドを使用して、sdlgendata からログ処理にデータを渡します。-vi オプションは、データが廃棄されてエラーが stdout に書き込まれることを指定します。データ生成を終了するには、CTRL+C またはその他の割り込み制御文字を押します。

```
sdlgendata logfile_set class -wait 5 | dsilog
```

```
\logfile_set class -s 10 -vi
```

上記のコマンドにより、次のようなデータが生成されます。

```
dsilog
I: 744996402 1.0000 2.0000 3.0000 4.0000 5.0000 6.0000 7.0000
I: 744996407 2.0000 3.0000 4.0000 5.0000 6.0000 7.0000 8.0000
I: 744996412 3.0000 4.0000 5.0000 6.0000 7.0000 8.0000 9.0000
I: 744996417 4.0000 5.0000 6.0000 7.0000 8.0000 9.0000 10.0000
I: 744996422 5.0000 6.0000 7.0000 8.0000 9.0000 10.0000 11.0000
I: 744996427 6.0000 7.0000 8.0000 9.0000 10.0000 11.0000 12.0000
I: 744996432 7.0000 8.0000 9.0000 10.000 11.000 12.0000 13.0000
I: 744996437 8.0000 9.0000 10.0000 11.0000 12.0000 13.0000 14.0000
```

また、dsilog の -vo オプションを使用して実際に実データを記録する前に、実データの入力および要約出力を検査できます。次のコマンドにより、vmstat は 5 秒間のインターバルで dsilog に渡され、10 秒間に要約されます。

```
->vmstat 5 | dsilog logfile_set class -s 10 -vo

dsilog

I: 744997230 0.0000 0.0000 21.0000 2158.0000 1603.0000 2.0000 2.0000

I: 744997235 0.0000 0.0000 24.0000 2341.0000 1514.0000 0.0000 0.0000

interval marker

L: 744997230 0.0000 0.0000 22.5000 2249.5000 1558.5000 1.0000 1.0000


I: 744997240 0.0000 0.0000 23.0000 2330.0000 1513.0000 0.0000 0.0000

I: 744997245 0.0000 0.0000 20.0000 2326.0000 1513.0000 0.0000 0.0000

interval marker

L: 744997240 0.0000 0.0000 21.5000 2328.0000 1513.0000 0.0000 0.0000


I: 744997250 0.0000 0.0000 22.0000 2326.0000 1513.0000 0.0000 0.0000

I: 744997255 0.0000 0.0000 22.0000 2303.0000 1513.0000 0.0000 0.0000

interval marker

L: 744997250 0.0000 0.0000 22.0000 2314.5000 1513.0000 0.0000 0.0000


I: 744997260 0.0000 0.0000 22.0000 2303.0000 1512.0000 0.0000 0.0000

I: 744997265 0.0000 0.0000 28.0000 2917.0000 1089.0000 9.0000 33.0000

interval marker

L: 744997260 0.0000 0.0000 25.0000 2610.0000 1300.5000 4.5000 16.5000


I: 744997270 0.0000 0.0000 28.0000 2887.0000 1011.0000 3.0000 9.0000

I: 744997275 0.0000 0.0000 27.0000 3128.0000 763.0000 8.0000 6.0000

interval marker

L: 744997270 0.0000 0.0000 27.5000 3007.5000 887.0000 5.5000 12.5000
```

dsilog-vo オプションを使用すると、古いデータのファイルもテストに使用できます。ただし、データに独自の UNIX タイムスタンプ (1/1/70 00:00:00 からの秒数) が含まれている場合に限ります。古いデータのファイルを使用するには、次のようなコマンドを入力します。

```
dsilog -timestamp -vo <oldfile>
```

# フォーマット ファイルの作成

次の場合、フォーマット ファイルを作成して、データ入力をクラスの仕様にマップします。

- データ入力が、クラスの仕様に含まれないデータを含んでいる場合
- 受信データのメトリックの順序が、クラスの仕様で指定した順序とは異なる場合

フォーマット ファイルは、vi または任意のテキスト エディタで作成できる ASCII テキスト ファイルです。dsilog の -f オプションを使用して、フォーマット ファイルの完全修飾名を指定します。

次のメトリックを開始するために、区切り記号 (デフォルトのスペース、または dsilog の -c オプションによるユーザー定義の記号) の後の最初の有効な文字を検索することによりログ処理は実行されるため、フォーマット ファイルはスキップするフィールドとスキップしないフィールドと関連するメトリック名のみをログ処理に通知します。

$numeric は、数値メトリック フィールドを 1 つスキップして次に進むようにログ処理に通知します。$any は、テキスト メトリック フィールドを 1 つスキップして次に進むようにログ処理に通知します。フォーマット ファイルは 100 フィールドまでに制限されることに注意してください。

たとえば、受信データ ストリームには次の情報が含まれています。

```
ABC 987 654 123 456
```

ここで、最初の数値フィールドのみをメトリック metric_1 に記録する場合、フォーマット ファイルは次のようになります。

```
$any metric_1
```

これは、最初の数値フィールドの情報のみを記録し、残りのデータを廃棄するようにログ処理に通知します。3 番目の数値フィールドの情報のみを記録する場合、フォーマット ファイルは次のようになります。

```
$any $numeric $numeric metric_1
```

4 つの数値データ項目すべてを逆の順序で記録する場合、フォーマット ファイルは次のようになります。

```
$any metric_4 metric_3 metric_2 metric_1
```

受信データ ストリームには次の情報が含まれています。

```
/users 15.9 3295 56.79% xdisk1 /dev/dsk/c0d0s*
```

最初のテキスト メトリックと最初の 2 つの数値フィールドを、それぞれ *text_1*、*num_1*、*num_2* と命名してメトリック フィールドに記録する場合、フォーマット ファイルは次のようになります。

```
text_1 num_1 num_2
```

これは、最初の 3 つのフィールドの情報のみを記録し、残りのデータを廃棄するようにログ処理に通知します。

すべてのデータを記録するが、3 番目のメトリックの後の「%」は廃棄する場合、フォーマット ファイルは次のようになります。

```
text_1 num_1 num_2 num_3 $any text_2 text_3
```

現在、数値フィールドを記録しており、「%」はテキスト フィールドと見なされるため、「%」の後のテキスト フィールドを正しく記録するには「%」をスキップする必要があります。

データ項目を異なる順序で記録する場合、フォーマット ファイルは次のようになります。

```
text_3 num_2 num_1 num_3 $any text_2 text_1
```

クラスの仕様で *text_1* の長さが 6 文字であると宣言されている場合、*text_3* の最初の 6 文字のみが記録されることに注意してください。*text_3* のすべてを最初の値として記録するには、クラスの仕様を変更して余分な領域を使用できるようにデータ ストリームを変更します。

# クラス仕様の変更

クラスの仕様ファイルを変更するには、次のようにログ ファイル セット全体を再度作成する必要があります。

1. `dsilog` プロセスを終了します。
2. データを保存する場合または古いデータと記録する新しいデータを統合する場合、UNIX タイムスタンプ オプションを使用して既存のログ ファイルからデータをエクスポートします。この方法については、この章で後述している「DSI データのエクスポート」を参照してください。
3. `sdlutil` を実行してログ ファイル セットを削除します。この方法については、この章で後述している「sdlutil によるデータの管理」を参照してください。
4. クラスの仕様ファイルを更新します。
5. `sdlcomp` を実行して、クラスの仕様を再コンパイルします。
6. 任意で `dsilog` で `-i` オプションを使用し、手順 2 でエクスポートした古いデータに統合します。場合によっては、`-f` *format_file* オプションを使用して、新しいデータと統合するデータを変更する必要があります。
7. `dsilog` を実行して、新しいクラスの仕様に基づく記録を開始します。
8. ログ ファイル セットの名前または位置を変更しない限り、`datasources` ファイルを更新する必要はありません。

# DSI データのエクスポート

DSI ログ ファイルからデータをエクスポートするには、Performance Collection Component の `export` プログラムの `export` 機能を使用します。`extract` プログラムを使用してデータをエクスポートする方法についての詳細は、『HP Operations エージェント for UNIX ユーザー マニュアル』5 章および6 章を参照してください。コマンド ラインの引数を使用して DSI データをエクスポートする例を次のページに示します。

DSI ログ ファイルからエクスポートできるクラスとメトリックを検出するには、いくつかの方法があります。この章で後述する「sdlutil を使用したデータの管理」で説明されているように、`sdlutil` を使用してこの情報をリストします。または、`extract guide` コマンドを使用して DSI ログ ファイルにクラスとメトリックをリストするエクスポート テンプレート ファイルを作成します。`vi` を使用して、ファイルを編集、命名、保存できます。『HP Operations エージェント for UNIX ユーザー マニュアル』5 章および6 章で記述した方法で、エクスポート テンプレート ファイルを使用してエクスポート フォーマットを指定します。

**注 :** DSI ログ ファイル データをエクスポートするには、ユーザーがログ ファイルの root または作成者

であることが必要です。

## DSI ログファイル データをエクスポートする extract プログラムの使用例

`extract -xp -l logfile_set -C class [options]`

`extract` コマンド ライン オプションを使用して、次の動作が実行できます。

- エクスポート出力ファイルの指定
- エクスポート対象の最初と最後のインターバルを開始する日時および終了する日時の設定
- 特定時間 (シフト) に限定したデータのエクスポート
- 特定曜日 (週末など) のデータの除外
- レポートのメトリック間に挿入する区切り記号の指定
- 到着するデータがないインターバルや表示された値が欠落データやヌル データ用であるインターバルに対して、見出しや空白レコードを表示するかどうかの選択
- エクスポートされた日付と時刻の表示 (UNIX フォーマットまたは日付フォーマット)
- 追加の要約レベルの設定

## Performance Manager でのデータの表示

Performance Manager で DSI ログファイルのデータを表示するには、DSI ログファイルを Performance Collection Component データソースとして構成する必要があります。データの記録を開始する前に、Performance Collection Component システムの `datasources` ファイルにデータを追加してデータソースを構成します。

Performance Manager を使用すると、DSI データの傾向の表示、監視、分析、比較、予測を集中処理できます。Performance Manager は、現在の問題や将来起こる可能性のある問題を識別するときに役立ちます。Performance Manager は、ユーザーの生産性に影響が生じる前に、問題解決に必要な情報を提供します。

# sdlutil によるデータの管理

DSI ログファイルのデータを管理するには、`sdlutil` プログラムを使用して次の作業のいずれかを実行します。

- 現在定義されているクラスとメトリック情報を `stdout` にリストします。出力をファイルにリダイレクトできます。
- クラスの詳しい統計値を `stdout` にリストします。
- リストしたすべてのメトリックに関するメトリックの記述を表示します。
- ログファイル セット内のファイルをリストします。
- ログファイル セットからクラスとデータを削除します。

- ログ ファイル セット内の情報からクラスの仕様を再度作成します。
- バージョン情報を表示します。

# 構文

`sdlutil logfile_set [option]`

| 変数とオプション | 定義 |
|---|---|
| `logfile_set` | クラスの仕様をコンパイルして作成したログ ファイル セットの名前です。 |
| `-classes` *classlist* | リストされたすべてのクラスに関するクラスの記述を提示します。クラスがリストされていない場合、すべてのクラスの記述が提示されます。*classlist* 内の項目はスペースで区切ります。 |
| `-stats` *classlist* | リストされたすべてのクラスの詳しい統計値を提示します。クラスがリストされていない場合、すべてのクラスの記述が提示されます。*classlist* 内の項目はスペースで区切ります。 |
| `-metrics` *metriclist* | *metriclist* 内のすべてのメトリックに関するメトリックの記述を提示します。クラスがリストされていない場合、すべてのクラスの記述が提示されます。*metriclist* 内の項目はスペースで区切ります。 |
| `-id` | ログ ファイルが使用する共有メモリ セグメント ID を表示します。 |
| `-files` | ログ ファイル セット内のすべてのファイルをリストします。 |
| `-rm all` | ログ ファイルからすべてのクラスとデータ、およびクラスに関するデータと共有メモリ ID を削除します。 |
| `-decomp` *classlist* | ログ ファイル セット内の情報からクラスの仕様を再度作成します。この結果は `stdout` に書き込まれます。ファイルを変更して再使用する場合、ファイルに結果をリダイレクトします。*classlist* 内の項目はスペースで区切ります。 |
| `-vers` | バージョン情報を表示します。 |
| `-?` | 構文の説明を表示します。 |

# データ ソースの統合例

データ ソース統合は極めて強力で柔軟な技術です。単純なものから非常に複雑なものまで DSI を実装できます。

この章では、DSI を使用して次の作業を使用する例を示します。

- **dsilog** スクリプトの記述
- vmstat データの記録
- sar データの記録
- who ワード カウントの記録

## dsilog スクリプトの記述

**dsilog** コードは、入力として、連続したデータ列の流れを扱うように考慮されています。入力されたデータ列は、各クラスの仕様指示による **dsilog** と、要求された要約間隔で記録される 1 つの要約されたデータ列によって要約されます。ログの中に書き込まれたタイムスタンプが、予想した要約割合 (レコード/時 ) に一致する場合、Performance Manager と perfalarm は最良な動作になります。これは **dsilog** が要約を行うことを許された場合、自動的に発生します。

スクリプトによっては各到着する入力列ごとに新しい **dsilog** プロセスが実行されます。これは、Performance Manager と perfalarm に関する問題を引き起こす原因になるために推奨しません。

- 問題となる **dsilog** スクリプト
- 推奨する **dsilog** スクリプト

### 例 1 - 問題となる dsilog スクリプト

次のスクリプトでは、新しいプロセス **dsilog** が各到着する入力列で実行されます。

```
while :
do
 feed_one_data_row | dsilog sdlname classname
 sleep 50
done
```

### 例 2 - 推奨する dsilog スクリプト

次のスクリプトでは、1 つの **dsilog** プロセスが、連続した入力データ列を受け付けます。feed_one_data_row は、連続的なデータ列を単一の **dsilog** プロセスに供給する機能として書かれています。

```
# Begin data feed function
feed_one_data_row()
 {
 while :
```

```
 do
# Perform whatever operations necessary to produce one row
# of data for feed to a dsilog process
 sleep 50
 done
 }
# End data feed function


# Script mainline code
 feed_one_data_row | dsilog sdlname classname
```

# vmstat データの記録

この例は、vmstat からレポートされる最初の 2 つの値を記録するために、デフォルト設定を使用してデータソース統合を準備する方法を示します。データソース統合処理機能の概要としてこの項を参照する、または各作業を実行してシステム上で等しい DSI ログファイルを作成することもできます。

データソース統合を実装するには、次の手順を実行することが必要です。

- クラスの仕様ファイルの作成
- クラスの仕様ファイルのコンパイル
- **dsilog** によるログ処理の開始

## クラス仕様のファイルの作成

クラスの仕様ファイルは、クラス内でメトリックとして記録する個別の数字と、クラス、すなわち受信データのセットを説明するために作成するテキスト ファイルです。ファイルは好みのテキスト エディタを使用して作成されます。ここで示すデータソース統合例のファイルは **/tmp/** ディレクトリに作成されます。

次の例では、VMSTAT_STATS の名前が付いたクラス内で最初の 2 つの vmstat 数値を記述するために必要なクラスの仕様ファイルを示します。このクラスに定義されたメトリックは 2 つであるため、ログプロセスは各 vmstat 出力レコードの残りを無視します。ファイル内の各ラインの内容は次の注釈ラインで説明されます。

```
CLASS VMSTAT_STATS = 10001;
 # Assigns a unique name and number to vmstat class data


 # The semicolon is required to terminate the class section
 # of the file.


METRICS
 # Indicates that everything that follows is a description
```

```
 # of a number (metric) to be logged.


RUN_Q_PROCS = 106;

 # Assigns a unique name and number to a single metric.

 # The semicolon is required to terminate each metric.


BLOCKED_PROCS = 107;

 # Assigns a unique name and number to another metric.

# The semicolon is required to terminate each metric.
```

## クラス仕 様 ファイルのコンパイル

sdlcomp を使 用してクラスの仕 様 ファイルをコンパイルする場 合 、ファイルを検 査して構 文 エラーを検 出します。エラーが検 出されない場 合 、sdlcomp はログ ファイル セットを作 成 または更 新して、クラスのデータを保 持します

クラスの仕 様 ファイルに付 けたファイル名 を使 用し、*logfile_set_name* の名 前 を指 定します。これにより、ログ ファイルに含 まれるデータの種 類 が容 易 に記 憶 できます。次 に示 すコマンド およびコンパイラ出 力 の例 で、**/tmp/vmstat.spec** はファイル名 として使 用され、/tmp/VMSTAT_DATA はログ ファイル セットに使 用されます。

```
-> sdlcomp /tmp/vmstat.spec /tmp/VMSTAT_DATA

sdlcomp X.01.04

Check class specification syntax.

CLASS VMSTAT_STATS = 10001;


METRICS

RUN_Q_PROCS = 106;

BLOCKED_PROCS = 107;
```

```
NOTE: Time stamp inserted as first metric by default.


Syntax check successful.


Update SDL VMSTAT_DATA.

Shared memory ID used by vmstat_data=219


Class VMSTAT_STATS successfully added to log file set.
```

この例では /tmp/ ディレクトリに **VMSTAT_DATA** ログ ファイル セットが作成され、データ ファイルにルート ファイルと記述ファイルが追加されます。ログ ファイル セットは記録されたデータをいつでも受信できます。クラスの仕様ファイルに構文エラーがある場合は障害を示すメッセージが表示され、ログ ファイル セットは作成されません。

## dsilog によるログ処理の開始

vmstat の出力を直接 **dsilog** ログ処理に渡します。次のコマンドを使用します。

```
vmstat 60 | dsilog /tmp/VMSTAT_DATA VMSTAT_STATS &
```

このコマンドは 60 秒ごとに vmstat を実行し、出力を VMSTAT_DATA ログ ファイル セット内の VMSTAT_STATS クラスに直接渡します。コマンドはバックグラウンドで実行されます。また、remsh を使用してリモート システムから vmstat を送信できます。

ログ処理の始めに、次のメッセージが生成されることに注意してください。

```
Metric null has invalid dataIgnore to end of line, metric value exceeds
maximum
```

このメッセージは、vmstat の出力で **dsilog** が記録できないヘッダー ラインの結果です。このメッセージが画面に表示されても **dsilog** は実行を続行し、最初の有効な入力ラインからデータの記録を開始します。

## データへのアクセス

sdlutil プログラムを使用して、次のようにクラスの内容をレポートできます。

```
sdlutil /tmp/VMSTAT_DATA -stats VMSTAT_STATS
```

> **注:** デフォルトでは、データは 5 分ごとに要約されて記録されます。

**extract** プログラムのコマンド ラインの引数を使用して、クラスからデータをエクスポートできます。たとえば、次のようになります。

```
extract -xp -l /tmp/VMSTAT_DATA -C VMSTAT_STATS -ut -f stdout
```

DSI データをエクスポートするには、ユーザーがログ ファイルの root または作 成 者 であることが必 要 です。

# 単 一 ファイルの sar データの記 録

この例 では、データを提 供 する標 準 の sar (システム アクティビティ レポート ) ユーティリティを使 用して、複 数 の DSI データ収 集をセットアップする方 法を示します。

システム ユーティリティを使 用する場 合 は、ユーティリティがデータをレポートする方 法を正 確 に把 握 することが重 要 です。たとえば、次 の 2 つの sar コマンド の違いに注 意してください。

```
sar -u 1 1

 HP-UX hpptc99 A.11.00 E 9000/855 04/10/99


 10:53:15 %usr %sys %wio %idle

10:53:16 2 7 6 85


sar -u 5 2

 HP-UX hpptc99 A.11.00 E 9000/855 04/10/99


 10:53:31 %usr %sys %wio %idle

 10:53:36 4 5 0 91 10:53:41 0 0 0 99


Average 2 2 0 95
```

上 記 の例 が示すように、`1` よりも大 きい反 復 値 を指 定 すると `sar` はインターバル間の平 均 値 を表 示します。この平 均 値 は、対 象になるかどうかにかかわらず `DSI` クラスの仕 様 ファイルとデータ変 換 に影 響 を与 えます。異 なる `UNIX` のプラットフォームで実 行 する場 合 は `sar` の出 力 、もしくはシステムのユーティリティの出 力 が異 なる場 合 があることに注 意してください。`DSI` クラスの仕 様 ファイルを作 成 する前 に、使 用 するユーティリティを十 分 に認 識 しておくことが必 要 です。

最 初 の例 では sar を使 用し、sar の -u オプションを介 して CPU の利 用 を監 視します。sar の man ページで示 すように、-u オプションはユーザー モード での実 行 時 間 (%usr)、システム モード での実 行 時 間 (%sys)、プロセスの一 部 がブロック I/O を待 つアイド ル時 間 (%wio)、それ以 外 のアイド ル時 間 (%idle) をレポートします。長 期 間 にわたって CPU のアクティビティを監 視 することを目 的 とするために、平 均 値 を表 示しない sar の形 式 を使 用します。

## クラス仕 様 のファイルの作 成

最 初 に、DSI クラスの仕 様 ファイルを作 成 します。次 の例 は、受 信 データを記 述 する場 合 に使 用 するクラスの仕 様 を示します。

```
# sar_u.spec

#
```

```
# sar -u class definition for HP systems.
#
# ==> 1 minute data; max 24 hours; indexed by hour; roll by day
#


CLASS sar_u = 1000
LABEL "sar -u data"
INDEX BY hour
MAX INDEXES 24
ROLL BY day
ACTION "./sar_u_roll $PT_START$ $PT_END$"
RECORDS PER HOUR 60
;

METRICS

hours_1 = 1001
LABEL "Collection Hour"
PRECISION 0
;

minutes_1 = 1002
LABEL "Collection Minute"
PRECISION 0
;

seconds_1 = 1003
LABEL "Collection Second"
PRECISION 0
;

user_cpu = 1004
```

```
LABEL "%user"
AVERAGED
MAXIMUM 100
PRECISION 0
;

sys_cpu = 1005
LABEL "%sys"
AVERAGED
MAXIMUM 100
PRECISION 0
;

wait_IO_cpu = 1006
LABEL "%wio"
AVERAGED
MAXIMUM 100
PRECISION 0
;

idle_cpu = 1007
LABEL "%idle"
AVERAGED
MAXIMUM 100
PRECISION 0
;
```

;

## クラス仕 様 ファイルのコンパイル

次 に、次 のコマンドを使 用してクラスの仕 様 ファイルをコンパイルします。

```
sdlcomp sar_u.spec sar_u_log
```

sar -u コマンド の出 力 は、システムのヘッダー ライン、空 白 ライン、オプション ヘッダーライン、データ ラインです。これらのラインはタイムスタンプで構 成 されていて、収 集 するデータが後 に続きます。対 象 となるのは最 終 ラインのみです。したがって、sar -u コマンド から出 力 の最 終 ラインのみを保 存 し、そのデータを DSI に送 信 する仕 様 が必 要 です。

**dsilog** プログラムは **stdin** からデータを受信することを予測します。ログ処理を開始するために、使用中の処理の出力を **dsilog** に渡します。ただし、コマンド ラインで使用できるのは 1 つのパイプ (|) のみです。2 つのパイプを使用した場合、UNIX バッファリングにより最初のコマンドの出力が遅延するためです。2 つ目のコマンドが実行されログ ファイルにデータが渡されるまで 8000 文字が書き込まれます。次のような結果が表示されます。

```
sar -u 60 1 | tail -1 | dsilog
```

この動作は有効でないため、DSI の入力ソースとして `fifo` を使用します。ただし、fifo には独自の問題があります。

次のスクリプトを使用していることを前提とします。

#!/bin/ksh sar_u_feed

# sar_u_feed script that provides sar -u data to DSI via

# a fifo(sar_u.fifo)

while :# (infinite loop)

do

# specify a one minute interval using tail to extract the

# last sar output record(contains the time stamp and data),

# saving the data to a file./usr/bin/sar -u 60 1 2>/tmp/dsierr | tail -1 >

/usr/tmp/sar_u_data

# Copy the sar data to the fifo that the dsilog process is

# reading.

cat /usr/tmp/sar_u_data > ./sar_u.fifo

done

残念なことに、現在ではこのスクリプトから希望する結果は得られません。これは、cat コマンドが fifo をオープンし、データ レコードを書き込んだ後、fifo をクローズするためです。fifo のクローズにより、ログに書き込むデータがないことが **dsilog** に示されるため、**dsilog** はこの 1 つのデータ レコードを書き込み終了します。fifo をオープン状態で「保持」するダミーの処理が必要です。そのためダミーの fifo が必要であり、入力のためにダミーの fifo をオープンし、出力のために sar_u.fifo をオープンする処理も必要となります。これにより、sar_u.fifo がオープン状態で保持されるため、**dsilog** が終了することはありません。

## DSI ログ処理の開始

次の手順を踏んで、sar -u データを **dsilog** に送信します。

1. fifo を 2 つ作 成します。1 つはダミーの fifo で、実 際 の入 力 fifo を「オープン状 態 に保 持 」するために使 用します。

   # Dummy fifo.

   mkfifo ./hold_open.fifo

   # Real input fifo for dsilog

   mkfifo ./sar_u.fifo

2. **-i** オプションを使 用して `dsilog` を開始し、`fifo` からの入 力を指定します。**sar** データ フィード (`sar_u_feed`) を開 始する前に、`dsilog` を開 始することが重 要 です。

   ```
   dsilog ./sar_u_log sar_u \
   -i ./sar_u.fifo &
   ```

3. ダミーの処 理を開 始し、入 力 `fifo` をオープン状 態 に保 持します。

   cat ./hold_open.fifo \

   > ./sar_u.fifo &

4. sar のデータ フィード のスクリプト (sar_u_feed) を開 始します。

   ./sar_u_feed &

5. sar_u_feed スクリプトはデータが抹 消 される、または **fifo** をオープン状 態 に保 持 する cat が抹 消されるまでデータを dsilog に供 給します。クラスの仕 様ファイルにより sar_u_log (最 大 24 時 間 分 ) は 1 時 間ごとにインデックスが付 けられ、翌 日 の最 初 に (1 日 ごとにロール) スクリプト sar_u_roll が実 行 されることが通 知 されます。

   !/bin/ksh sar_u_rol

   l#

   # Save parameters and current date in sar_u_log_roll_file.

   # (Example of adding comments/other data to the roll file).

   mydate=`date`

   echo "$# $0 $1 $2" >> ./sar_u_log_roll_file

   echo $mydate >> ./sar_u_log_roll_file

   extract -l ./sar_u_log -C sar_u -B $1 -E $2 -1 -f \

   stdout -xp >> ./sar_u_log_roll_file

6. ロール スクリプトは ASCII テキスト ファイルにロールアウトされるデータを保 存します。この ASCII テキスト ファイルでは、テキスト エディタによる検 査 やプリンタへの印 刷 が可 能 です。

# 複 数ファイルの sar データの記 録

CPU の利 用 のみでなく他 のデータも対 象 とする場 合 は、そのデータを記 述 するクラスの仕 様ファイルを 1 つ保 持 する、または各 オプションにクラスの仕 様ファイルを 1 つ保 持して、それらを 1 つのログ ファイル

セットにコンパイルできます。次に示す最初の例では、1 つのログファイルセットにコンパイルされた個別のクラスの仕様ファイルを示します。

次の例では、CPU の利用、バッファアクティビティ (sar -b)、システムコール (sar -c) を監視します。この方法でデータを記録する場合は、3 つのクラスの仕様ファイル、3 つの **dsilog** 処理、3 つの **dsilog** 入力 fifo、sar データを提供する 3 つのスクリプトが必要です。

## クラス仕様のファイルの作成

各オプションのクラスの仕様ファイルを次に示します。

```
# sar_u_mc.spec

#

# sar -u class definition for log files on HP systems.

#

# ==> 1 minute data; max 24 hours; indexed by hour; roll by day

#

CLASS sar_u = 1000

LABEL "sar -u data"

INDEX BY hour

MAX INDEXES 24

ROLL BY day

ACTION "./sar_u_mc_roll $PT_START$ $PT_END$"

RECORDS PER HOUR 60

;


METRICS

hours_1 = 1001

LABEL "Collection Hour"

PRECISION 0

;


minutes_1 = 1002

LABEL "Collection Minute"

PRECISION 0

;
```

```
seconds_1 = 1003
LABEL "Collection Second"
PRECISION 0
;
user_cpu = 1004
LABEL "%user"
AVERAGED
MAXIMUM 100
PRECISION 0
;

sys_cpu = 1005
LABEL "%sys"
AVERAGED
MAXIMUM 100
PRECISION 0
;

wait_IO_cpu = 1006
LABEL "%wio"
AVERAGED
MAXIMUM 100
PRECISION 0
;

idle_cpu = 1007
LABEL "%idle"
AVERAGED
MAXIMUM 100
PRECISION 0
;
```

```
# sar_b_mc.spec
#
# sar -b class definition for log files on HP systems.
#
# ==> 1 minute data; max 24 hours; indexed by hour; roll by day
#

CLASS sar_b = 2000
LABEL "sar -b data"
INDEX BY hour
MAX INDEXES 24
ROLL BY day
ACTION "./sar_b_mc_roll $PT_START$ $PT_END$"
RECORDS PER HOUR 60
;

METRICS
hours_2 = 2001
LABEL "Collection Hour"
PRECISION 0
;

minutes_2 = 2002
LABEL "Collection Minute"
PRECISION 0
;

seconds_2 = 2003
LABEL "Collection Second"
PRECISION 0
;

bread_per_sec = 2004
```

```
LABEL "bread/s"
PRECISION 0
;

lread_per_sec = 2005
LABEL "lread/s"
PRECISION 0
;

read_cache = 2006
LABEL "%rcache"
MAXIMUM 100
PRECISION 0
;

bwrit_per_sec = 2007
LABEL "bwrit/s"
PRECISION 0
;

lwrit_per_sec = 2008
LABEL "lwrit/s"
PRECISION 0
;

write_cache = 2009
LABEL "%wcache"
MAXIMUM 100
PRECISION 0
;

pread_per_sec = 2010
LABEL "pread/s"
```

```
PRECISION 0
;

pwrit_per_sec = 2011
LABEL "pwrit/s"
PRECISION 0
;

# sar_c_mc.spec
#
# sar -c class definition for log files on HP systems.
#
# ==> 1 minute data; max 24 hours; indexed by hour; roll by day
#

CLASS sar_c = 5000
LABEL "sar -c data"
INDEX BY hour
MAX INDEXES 24
ROLL BY day
ACTION "./sar_c_mc_roll $PT_START$ $PT_END$"
RECORDS PER HOUR 60
;

METRICS
hours_5 = 5001
LABEL "Collection Hour"
PRECISION 0
;

minutes_5 = 5002
LABEL "Collection Minute"
PRECISION 0
```

```
;

seconds_5 = 5003
LABEL "Collection Second"
PRECISION 0
;

scall_per_sec = 5004
LABEL "scall/s"
PRECISION 0
;

sread_per_sec = 5005
LABEL "sread/s"
PRECISION 0
;

swrit_per_sec = 5006
LABEL "swrit/s"
PRECISION 0
;

fork_per_sec = 5007
LABEL "fork/s"
PRECISION 2
;

exec_per_sec = 5008
LABEL "exec/s"
PRECISION 2
;

rchar_per_sec = 5009
```

```
LABEL "rchar"
PRECISION 0
;

wchar_per_sec = 5010
LABEL "wchar/s"
PRECISION 0
;
```

sar データを提供するために必要な 2 つの追加スクリプトを次に示します。

```
#!/bin/ksh
# sar_b_feed script that provides sar -b data to DSI via
# a fifo (sar_b.fifo)
while : # (infinite loop)
do

# specify a one minute interval using tail to extract the
# last sar output record(contains the time stamp and data),
 # saving the data to a file.

/usr/bin/sar -b 60 1 2>/tmp/dsierr | tail -1 &> \
/usr/tmp/sar_b_data


# Copy the sar data to the fifo that the dsilog process is reading.
cat /usr/tmp/sar_b_data > ./sar_b.fifo
done
#!/bin/ksh sar_c_feed

# sar_c_feed script that provides sar -c data to DSI via
# a fifo(sar_c.fifo)
while :# (infinite loop)
do
```

```
# specify a one minute interval using tail to extract the
 # last sar output record(contains the time stamp and data),
# saving the data to a file.

/usr/bin/sar -c 60 1 2>/tmp/dsierr | tail -1 > /usr/tmp/sar_c_data


# Copy the sar data to the fifo that the dsilog process is reading.
cat /usr/tmp/sar_c_data > ./sar_c.fifo
done
```

## 複数のクラスの仕様ファイルのコンパイル

3 つの仕様ファイルを 1 つのログ ファイル セットにコンパイルします。

```
sdlcomp ./sar_u_mc.spec sar_mc_log
sdlcomp ./sar_b_mc.spec sar_mc_log
sdlcomp ./sar_c_mc.spec sar_mc_log
```

## DSI ログ処理の開始

sar データについては、再度、次の手順を実行します。

1. fifo を 4 つ作成します。1 つはダミーの fifo で、実際の入力 `fifo` を「オープン状態に保持」するために使用します。

   ```
   # Dummy fifo.mkfifo ./hold_open.fifo
   # sar -u input fifo for dsilog.mkfifo ./sar_u.fifo
   # sar -b input fifo for dsilog.mkfifo ./sar_b.fifo
   # sar -c input fifo for dsilog.mkfifo ./sar_c.fifo
   ```

2. **-i** オプションを使用して `dsilog` を開始し、`fifo` からの入力を指定します。**sar** データ フィードを開始する前に `dsilog` を開始することが重要です。

   ```
   dsilog ./sar_mc_log sar_u \
   -i ./sar_u.fifo &
   ```

```
dsilog ./sar_mc_log sar_b \
-i ./sar_b.fifo &

dsilog ./sar_mc_log sar_c \
-i ./sar_c.fifo &
```

3. ダミーの処理を開始し、入力 `fifo` をオープン状態に保持します。

```
cat ./hold_open.fifo \
> ./sar_u.fifo &
cat ./hold_open.fifo \
 > ./sar_b.fifo &
cat ./hold_open.fifo \
> ./sar_c.fifo &
```

4. sar データフィードのスクリプトを開始します。

   ./sar_u_feed &

   ./sar_b_feed &

   ./sar_c_feed &

# 複数オプションの sar データの記録

sar を使用してデータを DSI に提供するための最後の例では、1 つの仕様ファイルを使用して複数の sar オプション (ubycwavm) からデータを定義しています。

```
# sar_ubycwavm.spec
#
# sar -ubycwavm class definition for HP systems.
#
# ==> 1 minute data; max 24 hours; indexed by hour; roll by day#

CLASS sar_ubycwavm = 1000
LABEL "sar -ubycwavm data"
INDEX BY hour
MAX INDEXES 24ROLL BY day
ACTION "./sar_ubycwavm_roll $PT_START$ $PT_END$"
RECORDS PER HOUR 60
```

```
;

METRICShours = 1001
LABEL "Collection Hour"
PRECISION 0
;

minutes = 1002
LABEL "Collection Minute"
PRECISION 0
;

seconds = 1003
LABEL "Collection Second"
PRECISION 0
;

user_cpu = 1004
LABEL "%user"
AVERAGED
MAXIMUM 100
PRECISION 0
;

sys_cpu = 1005
LABEL "%sys"
AVERAGED
MAXIMUM 100
PRECISION 0
;

wait_IO_cpu = 1006
LABEL "%wio"
```

```
AVERAGED
MAXIMUM 100
PRECISION 0
;

idle_cpu = 1007
LABEL "%idle"
AVERAGED
MAXIMUM 100
PRECISION 0
;

bread_per_sec = 1008
LABEL "bread/s"
PRECISION 0
;

lread_per_sec = 1009
LABEL "lread/s"
PRECISION 0
;

read_cache = 1010
LABEL "%rcache"
MAXIMUM 100
PRECISION 0
;

bwrit_per_sec = 1011
LABEL "bwrit/s"
PRECISION 0
;
```

```
lwrit_per_sec = 1012
LABEL "lwrit/s"
PRECISION 0
;

write_cache = 1013
LABEL "%wcache"
MAXIMUM 100
PRECISION 0
;

pread_per_sec = 1014
LABEL "pread/s"
PRECISION 0
;

pwrit_per_sec = 1015
LABEL "pwrit/s"
PRECISION 0
;

rawch = 1016
LABEL "rawch/s"
PRECISION 0
;

canch = 1017
LABEL "canch/s"
PRECISION 0
;

outch = 1018
LABEL "outch/s"
```

```
PRECISION 0
;

rcvin = 1019
LABEL "rcvin/s"
PRECISION 0
;

xmtin = 1020
LABEL "xmtin/s"
PRECISION 0
;

mdmin = 1021
LABEL "mdmin/s"
PRECISION 0
;

scall_per_sec = 1022
LABEL "scall/s"
PRECISION 0
;

sread_per_sec = 1023
LABEL "sread/s"
PRECISION 0
;

swrit_per_sec = 1024
LABEL "swrit/s"
PRECISION 0
;
```

```
fork_per_sec = 1025
LABEL "fork/s"
PRECISION 2
;

exec_per_sec = 1026
LABEL "exec/s"
PRECISION 2
;

rchar_per_sec = 1027
LABEL "rchar/s"
PRECISION 0
;

wchar_per_sec = 1028
LABEL "wchar/s"
PRECISION 0
;

swpin = 1029
LABEL "swpin/s"
PRECISION 2
;

bswin = 1030
LABEL "bswin/s"
PRECISION 1
;

swpot = 1031
LABEL "swpot/s"
PRECISION 2
```

```
;

bswot = 1032
LABEL "bswot/s"
PRECISION 1
;

blks = 1033
LABEL "pswch/s"
PRECISION 0
;

iget_per_sec = 1034
LABEL "iget/s"
PRECISION 0
;

namei_per_sec = 103
5LABEL "namei/s"
PRECISION 0
;

dirbk_per_sec = 1036
LABEL "dirbk/s"
PRECISION 0
;

num_proc = 1037
LABEL "num proc"
PRECISION 0
;

proc_tbl_size = 1038
```

```
LABEL "proc tbl size"
PRECISION 0
;

proc_ov = 1039
LABEL "proc ov"
PRECISION 0
;

num_inode = 1040
LABEL "num inode"
PRECISION 0
;

inode_tbl_sz = 1041
LABEL "inode tbl sz"
PRECISION 0
;

inode_ov = 1042
LABEL "inode ov"
PRECISION 0
;

num_file = 1043
LABEL "num file"
PRECISION 0
;

file_tbl_sz = 1044
LABEL "file tbl sz"
PRECISION 0
;
```

```
file_ov = 1045
LABEL "file ov"
PRECISION 0
;

msg_per_sec = 1046
LABEL "msg/s"
PRECISION 2
;

LABEL "sema/s"
PRECISION 2
;
```

この時点で、生成された出力を検査する必要があります。

```
sar -ubycwavm 1 1:
HP-UX hpptc16 A.09.00 E 9000/855 04/11/95

12:01:41 %usr %sys %wio %idle
bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s
pwrit/s
 rawch/s canch/s outch/s cvin/s xmtin/s mdmin/s
 scall/s sread/s swrit/s fork/s exec/s rchar/s wchar/s
swpin/s bswin/s swpot/s bswot/s pswch/s
 iget/s namei/s dirbk/s
text-sz ov proc-sz ov inod-sz ov file-sz ov
msg/s sema/s

12:01:42    22 48 30 0          0
342 100     33 81 59  0    0
 0 0 470      0 0 0
 801 127      71  1.00    1.00 975872 272384
```

0.00 0.0 0.00 0.0 251

28 215 107

N/A N/A 131/532 0 639/644 0 358/1141 0

40.00 0.00

この出力は、ヘッダーとデータに複数ラインが追加された sar -u 出力に類似しています。再度、末尾を使用してデータラインを抽出しますが、このデータラインを「1 つの」データレコードとして **dsilog** に提供する必要があります。次のスクリプトはデータを取り込み、tr (変換文字) ユーティリティを使用して改行を「削除」します。これは改行が **dsilog** により入力データの 1 行と見なされるためです。

```
#!/bin/ksh Sar_ubycwavm_feed#

 Script that provides sar data to DSI via a fifo(sar_data.fifo)

while :# (infinite loop)

do


# specify a one minute interval using tail to extract the

 # last sar output records (contains the time stamp and data)

# and pipe that data to tr to strip the new lines converting

 # the eight lines of output to one line of output.

/usr/bin/sar -ubycwavm 60 1 2>/tmp/dsierr | tail -8 | \

tr "\012" " " > /usr/tmp/sar_data


# Copy the sar data to the fifo that the dsilog process is reading.

cat /usr/tmp/sar_data > ./sar_data.fifo


# Print a newline on the fifo so that DSI knows that this is

# the end of the input record.


print "\012" > ./sar_data.fifo

done
```

順番に処理を実行すると、最初に示した sar -u の例は上記の結果となります。ただし、ログファイルセット名、クラス名、fifo 名 (sar_ubycwavm.fifo)、sar データを提供する上記にリストしたスクリプトは除きます。

# システム ユーザー数の記録

次の例では、who を使用してシステム ユーザー数を監視します。再度、クラスの仕様ファイルから開始します。

```
# who_wc.spec

#

# who word count DSI spec file

#


CLASS who_metrics = 150

LABEL "who wc data"

INDEX BY hour

MAX INDEXES 120

ROLL BY hour

RECORDS PER HOUR 60

;


METRICSwho_wc = 151

label "who wc"averaged

maximum 1000

precision 0

;
```

仕様ファイルをコンパイルして、ログ ファイルを作成します。

```
sdlcomp ./who_wc.spec ./who_wc_log.
```

sar と違って、who ではインターバルや反復値を指定できないため、最小限にインターバル制御を提供するスクリプトを作成します。

```
#!/bin/ksh who_data_feed

while :

do


 # sleep for one minute (this should correspond with the

 # RECORDS PER HOUR clause in the specification file).
```

```
sleep 60


 # Pipe the output of who into wc to count

 # the number of users on the system.


who | wc -l > /usr/tmp/who_data

# copy the data record to the pipe being read by dsilog.


cat /usr/tmp/who_data > ./who.fifo

done
```

データを dsilog に提供するために **fifo** とスクリプトが必要となるため、再度、次の手順を実行します。

1. fifo を 2 つ作成します。1 つはダミーの fifo で、実際の入力 fifo を「オープン状態に保持」するために使用します。

   .# Dummy fifo.

   mkfifo ./hold_open.fifo

   # Real input fifo for dsilog.

   mkfifo ./who.fifo

2. **-i** オプションを使用して dsilog を開始し、fifo からの入力を指定します。**who** データフィードを開始する前に、`dsilog` を開始することが重要です。

   dsilog ./who_wc_log who_metrics \-i ./who.fifo &

3. ダミーの処理を開始し、入力 fifo をオープン状態に保持します。

   cat ./hold_open.fifo \> ./who.fifo &

4. `who` のデータフィードのスクリプト (who_data_feed) を開始します。

   ./who_data_feed &

# エラー メッセージ

DSI エラー メッセージには、クラスの仕 様 、dsilog ログ処 理 、一 般 の 3 つのタイプがあります。

- クラスの仕 様エラー メッセージのフォーマットは、接 頭 辞 SDL +「メッセージ番 号 」です。
- dsilog ログ処 理メッセージのフォーマットは、接 頭 辞 DSILOG +「メッセージ番 号 」です。
- 一 般エラー メッセージは、その他 の作 業と同 様 、上 記 のいずれかにより生 成されます。これらのメッセージには、マイナス符 号 (-) の接 頭 辞とメッセージ番 号 が付きます。

DSI エラー メッセージは、この章 に記 載されています。SDL エラー メッセージと DSILOG エラー メッセージは、数 値 順 に、エラー状 態 から回 復するための対 応 方 法 と共 に記 載されています。一 般エラー メッセージは内 容 の説 明であり、回 復するための対 応 方 法 は記 載されていません。

## SDL エラー メッセージ

SDL エラー メッセージは Self Describing Logfile クラス仕 様 のエラー メッセージで、SDL<*message number*> というフォーマットで表します。

## DSILOG エラー メッセージ

DSILOG エラー メッセージは dsilog ログ処 理メッセージで、

DSILOG<*メッセージ番 号*> というフォーマットで表します。

# 一般エラー メッセージ

| エラー | 説明 |
|---|---|
| -3 | max-class で許可されている数より多くのクラスを追加しようとしています。 |
| -5 | クラス データを含むファイルをオープンできませんでした。 |
| -6 | ファイルを読み取れませんでした。 |
| -7 | ファイルに書き込めませんでした。 |
| -9 | 書き込みアクセスが要求されていないときに、ログ ファイルへの書き込みを行おうとしています。 |
| -11 | クラスへのポインタが見つかりませんでした。 |
| -13 | ファイルまたはデータ構造が初期化されていません。 |
| -14 | クラスの記述ファイルを読み取れませんでした。 |
| -15 | クラスの記述ファイルに書き込めませんでした。 |
| -16 | クラスの定義に必要なすべてのメトリックが、メトリックの記述クラスにありませんでした。 |
| -17 | ログ ファイル セット内のファイルのパス名の長さが、1024 文字を超えています。 |
| -18 | クラス名の長さが、20 文字を超えています。 |
| -19 | ファイルが、ログ ファイル セットのルート ファイルではありません。 |
| -20 | ファイルが、lod ファイル セットの一部ではありません。 |
| -21 | 現在のソフトウェアは、ログ ファイル セットにアクセスできません。 |
| -22 | 共有メモリ セグメントまたは ID が得られませんでした。 |
| -23 | 共有メモリ セグメントに結び付けられませんでした。 |
| -24 | ログ ファイル セットをオープンできません。 |
| -25 | 現在の作業ディレクトリを特定できませんでした。 |
| -26 | クラス データ ファイルからクラス ヘッダーを読み取れませんでした。 |
| -27 | ログ ファイル セット内のファイルをオープンできませんでした。 |
| -28 | データ クラスをオープンできませんでした。 |
| -29 | Lseek を実行できませんでした。 |
| -30 | ログ ファイルから読み取れませんでした。 |
| -31 | ログ ファイルに書き込めませんでした。 |

| | |
|---|---|
| -32 | 削除できませんでした。 |
| -33 | shmctl (REM_ID) を実行できませんでした。 |
| -34 | ログファイルセットが不完全です。ルートファイルまたは記述ファイルがありません。 |
| -35 | クラスを追加するターゲットログファイルが、現在のログファイルセットにありません。 |

# トランザクション追跡の紹介

この章では、次の内容について説明します。

- パフォーマンス管理の改善
- リアルタイムでの注文処理のシナリオ
- トランザクション データの監視

## パフォーマンス管理の改善

HP Operations エージェントおよび HP GlancePlus のトランザクション追跡機能によって、システム性能を管理する能力を改善できます。

分散した重要度の高いビジネス アプリケーションの数が増加するにつれて、アプリケーション マネージャおよびシステム マネージャはそれらの分散した情報技術 (IT: Information Technology) がどのように実行されているかを知るために、次のようなより多くの情報を必要とします。

- アプリケーションは応答を停止しているのか。
- アプリケーション応答時間は問題にならないか。
- サービスレベルの目標 (SLO) を達成しているか。

Performance Collection Component および GlancePlus のトランザクション追跡機能は、IT マネージャが、ビジネストランザクションを遂行するクライアント / サーバー IT 環境の終端間を管理しやすいように構築することを可能にします。Performance Collection Component により、ビジネストランザクションがどのようなものかを知り、ビジネスの遂行に重要なトランザクション データを定義し、収集できます。

標準的なアプリケーション応答測定 (ARM: Application Response Measurement) API コールがアプリケーションに組み込まれている場合、これらの製品はマルチベンダ プラットフォーム間での広範囲な処理内容の追跡および終端間の管理機能を提供します。

## トランザクション追跡の利点

- トランザクションの開始から終了までの経過時間をクライアントの視点で提供します。
- トランザクション データを提供します。
- サービスレベル協定 (SLA: Service Level Agreement) の管理を支援します。

これらの内容は、この後で詳細に説明します。

### ユーザーの視点から見たトランザクション時間

トランザクション追跡は、トランザクションの開始から終了までの経過時間をエンド ユーザーの視点で提供します。IT 環境でトランザクション追跡を使用すると、次の利点があります。

- 各トランザクションの実行回数を正確に追跡できます。
- 現在のような概算の時間ではなく、トランザクションが完了するまでの詳細な所要時間がわかります。
- トランザクション時間とシステムリソースの利用を関連させることができます。
- 容量立案、パフォーマンス管理、アカウンティング、課金などのシステム管理アプリケーションのデータを利用できます。
- システムとネットワークリソースの抽象的な定義によって実際の作業を表すのではなく、具体的な作業単位 (トランザクション) に基づいた、アプリケーションの最適化と詳細なパフォーマンストラブルシューティングが可能になります。

## トランザクション データ

各ビジネストランザクションの開始と終了をマークするために、アプリケーション応答測定 (ARM) API コールをアプリケーションに挿入すると、次のリソースおよびパフォーマンスの監視ツールを使用して、トランザクション データを監視できます。

- Performance Collection Component は、トランザクション データのアラームの検知、記録、報告に必要な登録機能を提供します。トランザクション データは Performance Manager、Glance で表示することも、Performance Collection Component のログファイルから、スプレッドシートやレポーティングツールによりアクセスできるファイルにデータをエクスポートして表示することもできます。
- Performance Manager によりグラフ化されたパフォーマンス データは、短期的なトラブルシュートや、長期的な傾向や分析などの調査に使用します。
- Glance は、システムおよびトランザクションの状態を監視するための詳細なリアルタイム データを逐次表示します。
- Performance Manager、Glance または HP Operations Manager のメッセージ ブラウザは、サービスレベルのコンプライアンス関するアラームを監視します。

各トランザクションのメトリックについては、22 章「トランザクション メトリック」を参照してください。

## サービス レベルの目標

サービスレベルの目標 (SLO: Service Level Objective) は、ビジネス アプリケーションのユーザーから要求される規定されたサービスレベルから算出されます。SLO は、一般的にサービスレベル協定 (SLA: Service Level Agreement) の作成に基づくものです。SLO からは、ビジネス アプリケーション ユーザーに対して合意したサービスレベルが満たされているかどうかを判断するために、IT リソース マネージャが収集、監視、保存、報告する必要がある実際のメトリックが作成されます。

SLO は、簡単なトランザクションの応答時間を監視するような単純な場合もありますが、システムの可用性の追跡のように複雑な場合もあります。

# リアルタイムでの注文処理のシナリオ

大勢の電話オペレータが、さまざまな商品の注文を視聴者から受けるテレビショッピングを想像します。この企業は、注文の情報を入力し、商品の納期をチェックし、在庫のインベントリを更新するために、コンピュータ プログラムを使用すると仮定します。この架空の企業を例に、顧客との契約および SLO を達成するために、トランザクション追跡がどのように役立つかを説明します。

リソース マネージャは、重要な作業、顧客満足度の要因、生産性の要因、最大応答時間に基づき、顧客に提供するサービスレベルを決定できます。

23 章「トランザクション追跡の例」には、トランザクション データを Performance Collection Component および Glance で監視できるように、ARM API コールを注文処理アプリケーションに挿入する方法の擬似コード例が示されています。

## リアルタイム注文処理の必要事項

上記のリアルタイム注文処理例において SLO を達成するために、リソース マネージャは次の重要な作業を完了するために必要な時間を監視する必要があります。

- 注文情報の入力
- 商品の納期の問い合わせ
- 在庫のインベントリの更新

ここで、顧客にとっての重要な顧客満足の要素は、オペレータが注文をどれくらい迅速に処理できるかです。

企業にとっての重要な生産性の要素は、オペレータが 1 時間に処理できる注文の数です。

顧客満足および生産性の目標を達成するには、設定した SLO のコンプライアンスのために、インベントリ データベースへのアクセス、インベントリの調整、レコードの書き戻しの各トランザクションの応答時間を監視する必要があります。たとえば、リソース マネージャがトランザクションの 90 パーセントが 5 秒以内で終了するように、このアプリケーションの SLO を設定している場合などです。

## 注文処理アプリケーションの準備

インベントリの応答およびインベントリの更新のトランザクションを作成するために、ARM API コールを注文処理アプリケーションに挿入できます。アプリケーション プログラマーは、アプリケーションをコンパイルする前に、ARM API コールをアプリケーションに挿入する必要があります。さまざまなトランザクション監視を定義する ARM API コールを含む注文処理プログラムの例 (擬似コードに記載) は、23 章「トランザクション追跡の例」を参照してください。

ARM API コールをアプリケーションに組み込む詳細は、『アプリケーション応答測定 2.0 API ガイド』を参照してください。

# トランザクション データの監視

ARM API コールが組み込まれているアプリケーションをシステムにインストールして実行すると、Performance Collection Component、GlancePlus、または Performance Manager でトランザクション データを監視できます。

**Performance Collection Component** での使用

Performance Collection Component を使用すると、指定したトランザクションのデータを収集および記録し、経時的に SLO の傾向を監視し、SLO を超過したときにアラームを生成できます。これらの傾向が特定されたら、トランザクションの量に基づいて IT コストを割り当てることができます。Performance Collection Component のアラームは、技術者のページャに通知するように構成できるため、問題を直ちに調査して解決することができます。詳細情報は、24 章「高度な機能」を参照してください。

Performance Manager でトランザクション データを表 示 するには、Performance Collection Component が必 要 です。

**Performance Manager での使 用**

Performance Manager は Performance Collection Component からアラームとトランザクション データを受 信します。たとえば、注 文 処 理 アプリケーションによる在 庫 確 認 に時 間 がかかりすぎる場合 、Performance Manager はアラームを受 信し、潜 在 的 な問 題として警 告をリソース マネージャのコンソールに送 信 するように Performance Collection Component を構 成 できます。

Performance Manager で、データ ソースの [Class List (クラス リスト )] ウィンド ウから **TRANSACTION** を選 び、さまざまなトランザクションの**トランザクション メトリックをグラフ化** できます。詳 細 は、Performance Manager のオンライン ヘルプを参 照 してください。

**GlancePlus での使 用**

GlancePlus を使 用 すると、最 新 のトランザクション応 答 時 間 と、設 定 した SLO 内 でトランザクションが実 行 されているかどうかを監 視 できます。GlancePlus は、トランザクションのパフォーマンスに影 響 を与えている可 能 性 のあるリソースのボトルネックの特 定 および解 決 に役 立 ちます。詳 細 は、GlancePlus ヘルプ メニューから利 用 できる GlancePlus オンライン ヘルプを参 照 してください。

# ARM の使 用 のガイドライン

ARM API をアプリケーションに組 み込 む場 合 、いくつかの注 意 点 があります。さらに、ARM データ収集 の特 徴 と制 限 を理 解 していれば、ARM の機 能 が組 み込 まれたアプリケーションの環 境 をより簡 単 に管 理 できます。これらを十 分 に理 解 していない場 合 に混 乱 が生 じることのある点 を次 に示 します。

1. ARM メトリックを取 得 するには、`ttd` と`midaemon` が動 作 している必 要 があります。Performance Collection Component の場 合 、ARM メトリックを記 録 するには `scope` コレクタが動 作 している必 要 があります。`ovpa start` スクリプトは、必 要 なプロセスをすべて起 動 します。同 様 に Glance は、`ttd` と`midaemon` が開 始 していない場 合 、それらを開 始 させます(19 章「トランザクション追 跡 デーモン (ttd)」を参 照 )。
2. 新 しく定 義 したトランザクション名 を取 得 するには、トランザクション構 成 ファイル `ttd.conf` を再 度 読 み込 みます(19 章「トランザクション構 成 ファイル (ttd.conf)」を参 照 )。
3. 新 規 または変 更 されたトランザクションの範 囲 とサービス レベルの目 標 (SLO) を取 得 するには、Performance Collection Component、ユーザー アプリケーション、`ttd` を再 起 動 する必 要 があります (19 章「新 しいアプリケーションの追 加 」を参 照 )。
4. Performance Collection Component では、ユーザー定 義 メトリック内 の文 字 列 は無 視 されます。最 初 の 6 つの非 文 字 列 ユーザー定 義 メトリックのみが記 録 されます(24 章「データ型 の使 用 方 法 」を参 照 )。
5. トランザクションのアラーム状 態 を指 定 する場 合 、トランザクション名 へのダッシュ (-) の使 用 には制 限 があります(20 章「アラーム」セクションの「Performance Collection Component の使 用 」を参 照 )。
6. Performance Collection Component は、アプリケーション名 およびトランザクション名 の最 初 の 60 文 字 のみを表 示 します(19 章「アプリケーション名 とトランザクション名 の指 定 」を参 照 )。
7. 組 み込 まれる固 有 のトランザクション名 の数 を制 限 します(20 章「固 有 のトランザクションの制 限 」を参 照 )。
8. エンド ユーザーから見 て、アプリケーションの実 行 に影 響 を与 えるような ARM API 関 数 コールを許 可 しません(19 章「ARM API コール ステータス リターン」を参 照 )。

9. リンクには、共 有 ライブラリを使 用します(「C Compiler Option Examples by Platform on page 378」のセクションを参 照 )。

# トランザクション追跡の動作

次に示す Performance Collection Component と GlancePlus のコンポーネントを一緒に使用すると、アプリケーション応答測定 (ARM: Application Response Measurement) コールが組み込まれたアプリケーションからのトランザクション データの定義や追跡に役立ちます。

- 測定インターフェイス デーモン (`midaemon`) は、Performance Collection Component、Performance Manager、GlancePlus によってアクセスおよびレポートされる情報のある共有メモリー セグメントで、トランザクション データを監視してレポートするデーモン プロセスです。また、HP-UX システムでは、`midaemon` はシステム パフォーマンス データも監視しています。
- 構成ファイル (`/var/opt/perf/ttd.conf`) は、ユーザーがトランザクションを定義し、各トランザクションで監視する情報を識別するために使用されます。
- トランザクション追跡デーモン (`ttd`) は、`midaemon`と共に、トランザクション構成ファイル (`ttd.conf`) からのトランザクション定義の読み込み、登録、同期を実行します。

## ARM 2.0 のサポート

ARM 2.0 は、以前の ARM バージョンのスーパーセットです。ARM 2.0 で提供される新しい機能は、ユーザー定義メトリック、トランザクション相関、ロギング エージェントです。Performance Collection Component および GlancePlus では、ユーザー定義メトリック、トランザクション相関はサポートされていますが、ロギング エージェントはサポートされていません。

ただし、アプリケーションへの組み込みをテストするために、ロギング エージェントを使用することはできます。ロギング エージェントのソース コード (`logagent.c`) は ARM 2.0 Software Developers Kit (SDK) に含まれています。これは、次の Web サイトから入手できます。

**http://regions.cmg.org/regions/cmgarmw**

ロギング エージェントの使い方については、『*アプリケーション応答測定 2.0 API ガイド*』を参照してください。

> **注 :** 『*アプリケーション応答測定 2.0 API ガイド*』では「ユーザー定義メトリック」ではなく、「アプリケーション定義メトリック」という用語で説明されています。

## ARM API コールのサポート

次に示す ARM API コールは、Performance Collection Component および GlancePlus でサポートされています。

| | |
|---|---|
| `arm_init()` | アプリケーションと (オプションとして) ユーザーを指定して登録します。 |
| `arm_getid()` | トランザクション クラスを指定して登録し、関連するトランザクション情報を提供します。ユーザー定義メトリックのコンテキストを定義します。 |
| `arm_start()` | 固有のトランザクション インスタンスの開始を通知します。 |

| | |
|---|---|
| arm_update() | 固有のトランザクション インスタンスの値を更新します。 |
| arm_stop() | 固有のトランザクション インスタンスの終了を送信します。 |
| arm_end() | アプリケーションの終了を送信します。 |

ARM API コールのアプリケーションへの組み込みやコールとパラメータの詳細については、現在の『*アプリケーション応答測定 2.0 API ガイド*』および *arm (3)* のマニュアル ページを参照してください。業務用アプリケーションでは、ARM API コールがアプリケーションにすでに組み込まれているかどうかを、その製品ドキュメントで確認してください。

必要なライブラリに関する重要な情報は、このマニュアルの後ろのTransaction Libraries on page 373を参照してください。

## arm_complete_transaction コール

HP の ARM エージェントは、ARM 2.0 API の標準の他に、arm_complete_transaction コールもサポートしています。このコールは ARM 標準への HP 特有の拡張であり、トランザクションの開始が arm_start コールで区切られなかった場合に、完了したトランザクションの終了をマークするために使用できます。arm_complete_transaction コールは、完了したトランザクション インスタンスの応答時間をパラメータとして使用します。

トランザクション インスタンスの終了を通知する他に、オプションのデータバッファには、トランザクションに関する追加情報を含めることができます。このコールとパラメータについての詳細情報に加え、このオプション データの情報については、*arm (3)* のマニュアル ページを参照してください。

## ARM 組み込みアプリケーションの例

組み込まれた ARM API コールの例については、/<InstallDir>/examples/arm/ ディレクトリにある armsample1.c、armsample2.c、armsample3.c、armsample4.c の ARM 組み込みアプリケーション、およびそのビルド スクリプト (Make.armsample) を参照してください。

- armsample1.c は簡単な標準 ARM API コールの使用例を示しています。
- armsample2.c も、簡単な標準 ARM API コールの使用例を示しています。これは、armsample1.c に似た構造を持ちますが、双方向の特性があります。
- armsample3.c は、ARM API のバージョン 2.0 で提供された、ユーザー定義メトリックとトランザクション相関係数を使用する方法の例です。この例は、サーバーとクライアントの両方がいくつかのトランザクションを実行するクライアント / サーバー アプリケーションをシミュレーションします(通常は、アプリケーション クライアントとサーバー コンポーネントは別々のプログラムに存在しますが、わかりやすくするために同じ場所に置かれています)。
  クライアント プロシージャはトランザクションを開始し、arm_start コールから ARM 相関係数を要求します。この相関係数はクライアントによって保存され、サーバーが arm_start をコールしたときに使用できるように、サーバーに渡されます。サーバーで実行されているパフォーマンス ツールは、この相関係数情報を使用して、このサーバーを使用している異なるクライアントを識別できます。

  また、このプログラムには、ユーザー定義のメトリック値を ARM API に渡すメカニズムが含まれています。これにより、パフォーマンス ツールの応答時間やサービスレベル情報がわかるだけではなく、アプリケーション自体にとって重要なデータを参照することもできます。たとえば、あるトランザクションが

異なるサイズの要求を処理していて、その要求のサイズをユーザー定義のメトリックで設定できる場合もあります。応答時間値が高い場合、長い応答時間が大きいサイズのトランザクション インスタンスに関係するかどうかを確認するために、このユーザー定義メトリックを使用できます。

- armsample4.c は、ARM コールのユーザー定義メトリックの使用例を提供します。arm_start、arm_update、arm_stop の各コールを使用して、異なるメトリック値を渡すことができます。または、arm_complete_transaction は、tran が start/stop コールで区切られない場合に使用できます。

## アプリケーション名とトランザクション名の指定

ARM の arm_init と arm_getid の API コール内のアプリケーション名とトランザクション名はそれぞれ最大 128 文字ですが、Performance Collection Component には最大 60 文字のみが表示されます。最初の 60 文字を超えた文字は表示されません。ただし、GlancePlus では 128 文字まで表示されます。

Performance Collection Component には、アプリケーション名およびトランザクション名が抽出またはエクスポートされたトランザクション データ内の表示方法に制限があります。これらの規則は、Performance Manager でアプリケーション名およびトランザクション名を表示するときにも適用されます。

アプリケーション名はトランザクション名より常に先行されます。たとえば、65 文字のアプリケーション名、および 40 文字のトランザクション名を持つトランザクション データをエクスポートすると、アプリケーション名のみが表示されます。アプリケーション名の最後の 5 文字は表示されません。

別の例としては、アプリケーション名が 32 文字で、トランザクション名が 40 文字の場合、Performance Collection Component はアプリケーション名をすべて表示しますが、トランザクション名は切り詰められ、合計で 60 文字が表示されます。アプリケーション名およびトランザクション名に 59 文字が割り当てられ、2 つの名前を区分する下線 (_) に 1 文字が割り当てられます。アプリケーション名「WarehouseInventoryApplication」およびトランザクション名「CallFromWestCoastElectronicSupplier」が Performance Collection Component または Performance Manager に表示される方法を次に示します。

```
WarehouseInventoryApplication_CallFromWestCoastElectronicSup
```

**注 :** Performance Manager でデータを表示する場合、アプリケーション名およびトランザクション名の 60 文字の組み合わせは固有である必要があります。

# トランザクション追跡デーモン (ttd)

トランザクション追跡デーモン (ttd) は、midaemon と共に ttd.conf からのトランザクション定義の読み込み、登録、同期を実行します。

ttd は、ovpa start コマンドで Performance Collection Component の scope データコレクタを起動すると開始します。ディスパッチされると、ttd はバックグラウンド モードで動作し、エラーは /var/opt/perf/status.ttd ファイルに書き込まれます。

また、トランザクションを処理し、これらのトランザクションに関連したパフォーマンス メトリックを収集するためには、midaemon を実行する必要があります (次のページを参照)。

**注意 :** ttd を停止しないことを強くお勧めします。

> ttd を停止させる必要がある場合、ttd とPerformance Collection Component のプロセスを再起動する前に、実行中の ARM 組み込みアプリケーションをすべて終了させる必要があります。ttd は、システム上で実行されている arm_init および arm_getid コールを取得するために、動作している必要があります。ttd が停止して再起動すると、これらのコールにより返されたトランザクション ID は繰り返されるため、ARM メトリックは無効になります。

Performance Collection Component のプロセスを起動して、プロセスが正確な順序で開始されたことを確認するには、ovpa スクリプトを使用します。ovpa stop では ttd をシャットダウンできません。任意のパフォーマンス ソフトウェアを再インストールするために ttd をシャットダウンする必要がある場合、/<InstallDir>/bin/ttd -k コマンドを使用します。ただし、Performance Collection Component を再インストールする以外では、ttd を停止させることはお勧めしません。

Performance Collection Component がシステム上にない場合、GlancePlus は midaemon を開始します。実行されていない場合、midaemon は midaemon が測定データの処理を開始する前に ttd を開始させます。

すべてのプログラム オプションについては、ttd のマニュアル ページを参照してください。

## ARM API コール ステータス リターン

トランザクションを登録するには、ttd プロセスが常に実行されている必要があります。ttd が何らかの理由で終了し、実行されていない場合、arm_init または arm_getid コールは「失敗」というリターン コードを返します。ttd がその後再起動すると、新しい arm_getid コールは、他のプログラムがすでに使用しているものと同じトランザクション ID を再登録する可能性があります。そうすると、無効なデータが記録されます。

ttd を停止して再起動すると、ARM 組み込みアプリケーションは、-2 (TT_TTDNOTRUNNING) の戻り値とARM API コールの EPIPEerrno エラーを受信し始めることがあります。アプリケーションが初めて起動すると、最初の ARM API コールでクライアント接続ハンドルが作成されます。このクライアント ハンドルにより、アプリケーションは ttd プロセスと通信することができます。ttd が終了するとこの接続も無効になるため、次にアプリケーションが ARM API コールの使用を試みると、TT_TTDNOTRUNNING の戻り値が送信されることがあります。このエラーは、たとえ他の ttd プロセスが実行されていても、前の ttd プロセスは実行されていないことを意味します (ARM API コールのリターンについては、*arm (3)* のマニュアル ページを参照してください)。

このエラーを回避するには、ttd が停止したら、ARM 組み込みアプリケーションを再起動する必要があります。はじめに、ARM 組み込みアプリケーションを終了させます。次に、(/<InstallDir>/bin/ovpa start または /<InstallDir>/bin/ttd を使用して) ttd を再起動し、その後にアプリケーションを再起動します。アプリケーションを再起動すると、アプリケーションと ttd プロセスの間に新しいクライアント接続ハンドルが作成されます。

midaemon にエラーがあっても、いくつかの ARM API コールはエラーを返しません。たとえば、midaemon がその共有メモリー セグメント内の領域を使い果たした場合です。パフォーマンス メトリック GBL_TT_OVERFLOW_COUNT は 0 より大きくなります。オーバーフローの状況が発生すると、(ttd 以外の) 実行中のパフォーマンス ツールをシャットダウンし、共有メモリー セグメント内でより多くの領域を指定するために、-smdvss オプションを使用して、midaemon を再起動することもできます。>(詳細は、*midaemon* のマニュアル ページを参照してください)。

ARM エラーが発生しても、アプリケーションが実行し続けられるようにアプリケーションを作成することをお勧めします。ARM の状態がプログラムの実行に影響しないようにします。

arm_getid コールによってトランザクションを ttd に登録することができるアクティブなクライアント プロセスの数は、maxfiles カーネル パラメータに制限されます。このパラメータは、オープンするファイル数をプロセスごとに制御します。各クライアント登録の要求により、ttd で RPC 接続用のソケット (オープンされた 1 つのファイル) が開かれます。クライアント アプリケーションが終了すると、ソケットは閉じられます。そのため、この制限は、arm_getid コールによってトランザクションを登録した、稼働中のクライアントの数にのみ影響します。この制限に達すると、ttd はクライアントの arm_getid 要求に対して TT_TTDNOTRUNNING を返します。maxfiles カーネル パラメータを増やすと、ttd にトランザクションを登録する、稼働中のアプリケーション数の上限値を上げることができます。

# 測定インターフェイス デーモン (midaemon)

測定インターフェイス デーモン (Measurement Interface Daemon: midaemon) は、システム パフォーマンス情報を継続的に収集する、負荷の小さいプロセスです。midaemon は、Performance Collection Component がトランザクション データを収集し、GlancePlus がトランザクション データを報告するために実行されている必要があります。scope または perfd のプロセスを実行するか GlancePlus を開始すると、midaemon は動作し始めます。

トランザクションを登録および追跡できるように、Performance Collection Component と GlancePlus では、midaemon と ttd の両方が実行されている必要があります。ovpa スクリプトは、mideamon を含む Performance Collection Component のプロセスを正しい順序で起動および終了させます。GlancePlus は mideamon が実行されていない場合にそれを起動します。midaemon は、ttd が実行されていない場合にそれを起動します。

midaemon の CPU オーバーヘッドについての情報は、このマニュアルの「「CPU オーバーヘッド 」(334 ページ)」を参照してください。

すべてのプログラム オプションについては、*midaemon* のマニュアル ページを参照してください。

# トランザクション構成ファイル (ttd.conf)

トランザクション構成ファイル /var/opt/perf/ttd.conf により、アプリケーション名、トランザクション名、パフォーマンス区分範囲、および各トランザクションに適用するサービスレベルの目標を定義できます。ttd は、各トランザクションを登録する方法を決定するために ttd.conf を読み込みます。

ttd.conf のカスタマイズはオプションです。Performance Collection Component に付属しているデフォルトの構成ファイルは、アプリケーションに組み込まれているすべてのトランザクションを監視します。

業務用アプリケーションを使用していて、どのようなトランザクションがアプリケーションに組み込まれているかがわからない場合、デフォルトの ttd.conf ファイルを使用して、いくつかのデータを収集します。次に、利用可能なトランザクションを知るために、そのデータを調べます。その後、ttd.conf を変更して、そのアプリケーションのトランザクション データの収集をカスタマイズできます。

## 新しいアプリケーションの追加

新しい ARM 組み込みアプリケーションを ttd.conf ファイル内のデフォルトの slo と tran=* 行の range 値でシステムに追加する場合、新しいトランザクションを組み込む操作は不要です(tran、range、slo の詳細は、「構成ファイルのキーワード 」のセクションを参照してください)。新しいトランザクションは自動的に取り出され、ttd.conf ファイル内の tran 行の slo および range の値が、新しいトランザクションに使用されます。

# 新しいトランザクションの追加

ttd.conf ファイルに追加を行ったら、次の手順を実行して、追加を有効にする必要があります。

- すべてのアプリケーションを終了する。
- ttd -hup -mi コマンドを root として実行する。

以上の操作により、ttd.conf ファイルが再度読み込まれ、新しいトランザクションが slo と range の値と共に ttd と midaemon に登録されます。再度読み込んでも、再読み込み前に ttd.conf ファイルにあったトランザクションの slo と range の値は変更されません。

# range 値または slo 値の変更

ttd.conf ファイルにある既存のトランザクションの slo と range の値を変更する場合、次を実行します。

- すべての ARM 組み込みアプリケーションを終了する。
- ovpa stop を使用して scope コレクタを終了する。
- Glance の使用をすべて停止する。
- ttd -k コマンドを発行して ttd を停止する。
- ttd.conf ファイルを変更した後、
- ovpa start を使用して、scope を再起動する。
- ARM 組み込みアプリケーションを再起動する。

# 構成ファイルのキーワード

構成ファイル /var/opt/perf/ttd.conf は、「構成ファイルのキーワード」(329ページ) に示すキーワードによって定義されるトランザクション属性に、トランザクション名を関連付けます。

**表 1:構成ファイルのキーワード**

| キーワード | 構文 | 使用方法 |
|---|---|---|
| tran | tran=*transaction_name* | 必要 |
| slo | slo=*sec* | オプション |
| range | range=*sec [,sec,...]* | オプション |

次に、これらのキーワードの詳細について説明します。

## tran

tran はトランザクション名の定義に使用します。この名前は、組み込みアプリケーション内の arm_getid API コールで定義されているトランザクションに対応する必要があります。オプションの属性 range または slo を指定する前に、tran キーワードを使用する必要があります。tran は構成ファイ

ルに必要な唯一のキーワードです。トランザクション名の末尾をアスタリスク (*) にすると、このエントリに対して登録要求が行われたときに、ワイルドカード パターン マッチが行われます。トランザクション名にはダッシュ (-) を使用できます。ただし、トランザクション名にスペースは使用できません。

トランザクション名は最大 128 文字まで使用できます。ただし、最初の 60 文字だけが Performance Collection Component に表示されます。GlancePlus は専用の画面で 128 文字を表示できます。

デフォルトの `ttd.conf` ファイルには、いくつかのエントリが含まれています。最初のエントリは、ARM API コールに組み込まれている Performance Collection Component のデータコレクタ `scope` で使用されるトランザクションを定義します。また、このファイルにはエントリ `tran=*` が含まれています。これは、ARM API またはトランザクション追跡 API コールに組み込まれているアプリケーション内のその他のトランザクションをすべて登録します。

## range

`range` はトランザクション パフォーマンス区分範囲を指定するために使用します。パフォーマンス区分範囲により、各トランザクションが完了するまでの各所用時間の区分と、各所用時間内にトランザクションが成功した回数を知ることができます。定義された範囲は GlancePlus の [トランザクション追跡] ウィンドウに表示されます。

`sec` に入力されたそれぞれの値は、その範囲のトランザクションの上限を秒単位で表します。値は小数点以下最大 6 桁で、整数または実数です。HP-UX では、1 マイクロ秒 (0.000001秒) の精度が許容されます。ただし、他のプラットフォームでの精度は 10 ミリ秒 (0.01 秒) です。したがって、許容されるのは小数点以下 2 桁のみです。

最大 10 個の範囲を各トランザクションごとに定義できます。
ユーザーは最大 9 個の範囲を指定できます。残り 1 個の範囲は、ユーザーが定義した最大の範囲を超えるトランザクション データを収集する、オーバーフロー範囲として確保されています。9 個以上の範囲を指定した場合、最初の 9 個が使用され、その後は無視されます。

9 個未満の範囲を指定した場合、最初の非指定範囲がオーバーフロー範囲になります。他の残りの非指定範囲は使用されません。非指定範囲のメトリックは、0.000 としてレポートされます。対応する最初の非指定カウント メトリックはオーバーフロー カウントになります。残りの非指定範囲のメトリックは常にゼロ (0) です。

範囲は昇順で定義する必要があります (この章の後の例を参照)。

## slo

`slo` は、パフォーマンス サービスレベル協定 (SLA) を監視するために必要なサービスレベルの目標 (SLO) を、秒単位で指定するために使用します。

`range` キーワードと同じように、値は小数点以下最大 6 桁の数字で、整数または実数です。HP-UX では、1 マイクロ秒 (0.000001秒) の精度が許容されます。ただし、他のプラットフォームでの精度は 10 ミリ秒 (0.01 秒) です。したがって、許容されるのは小数点以下 2 桁のみです。

HP-UX で、マイクロ秒でトランザクションを並べ替えることができても、トランザクション時間は 100 マイクロ秒単位で表示されることに注意してください。

# 構成ファイルのフォーマット

ttd.conf ファイルには、一般的なトランザクションとアプリケーション固有のトランザクションの 2 種類のエントリが含まれています。

一般的なトランザクションは、アプリケーションが定義される前に、ttd.conf ファイルで定義する必要があります。これらのトランザクションは、定義されるすべてのアプリケーションに関係しています。デフォルトの ttd.conf ファイルには一般的なトランザクション エントリが 1 つと、ARM API コールに組み込まれた scope コレクタのエントリが複数含まれます。

```
tran=* range=0.5, 1, 2, 3, 5, 10, 30, 120, 300 slo=5.0
```

オプションとして、各アプリケーションはトランザクション名の固有のセットを持つことができます。これらのトランザクションはそのアプリケーションのみに関係します。指定するアプリケーション名は、組み込みアプリケーション内の arm_init API コールに定義されているアプリケーション名に対応する必要があります。アプリケーション固有のエントリの各グループは、次に示すように、[ ] で囲まれたアプリケーションの名前から始まる必要があります。たとえば、次のようになります。

```
[AccountRec]

tran=acctOne range=0.01, 0.03, 0.05
```

アプリケーション名は最大 128 文字まで使用できます。ただし、最初の 60 文字だけが Performance Collection Component に表示されます。Glance は、専用の画面で 128 文字を表示できます。

一般的なトランザクションと同じ名前のトランザクションがある場合、アプリケーションに記述されたトランザクションが使用されます。

たとえば、次のようになります。

```
tran=abc range=0.01, 0.03, 0.05 slo=0.10

tran=xyz range=0.02, 0.04, 0.06 slo=0.08

tran=t* range=0.01, 0.02, 0.03


[AccountRec}

tran=acctOne range=0.04, 0.06, 0.08

tran=acctTwo range=0.1, 0.2

tran=t* range=0.03, 0.5


[AccountPay]

[GenLedg]

tran=GenLedgOne range=0.01
```

上記の例では、最初の 3 つのトランザクションは、指定される 3 つのアプリケーションのすべてに当てはまります。

アプリケーション [AccountRec] には、トランザクション acctOne、acctTwo、abc、xyz、t* があります。一般的なトランザクション セット内のエントリのうちの 1 つには、「t*」という名前のワイルドカードトランザクションがあります。この場合、AccountRec アプリケーションの「t*」トランザクション名が使用され、一般的なトランザクション セット内のトランザクション名は無視されます。

アプリケーション [AccountPay] には一般的なトランザクション セットからのトランザクションのみがあります。

アプリケーション [GenLedg] には、トランザクション GenLedgOne、abc、xyz、t* があります。

トランザクション名の順序によって、アプリケーションで違いが生じることはありません。

アプリケーション名およびトランザクション名のその他の情報は、この章の「「アプリケーション名とトランザクション名の指定」(326ページ)」を参照してください。

# 構成ファイル例

**例 1**

```
tran=* range=0.5,1,2,3,5,10,30,12,30 slo=5.0
```

「*」エントリは、登録されているトランザクション名とエントリが一致していない場合に、デフォルト値として使用されます。これらのデフォルト値は、各システム上で「*」エントリを修正して変更できます。「*」エントリが見当たらない場合、上記の「*」エントリに割り当てられている初期値パラメータと一致する登録パラメータのデフォルトのセットが使用されます。

**例 2**

```
[MANufactr]

tran=MFG01 range=1,2,3,4,5,10 slo=3.0

tran=MFG02 range=1,2.2,3.3,4.0,5.5,10 slo=4.5

tran=MFG03

tran=MFG04 range=1,2.2,3.3,4.0,5.5,10
```

MANufctr アプリケーションのトランザクションです。MFG01、MFG02、MFG04 は、それぞれ固有のパラメータを使用します。MFG03 トランザクションは、時間区分およびサービスレベルの目標を追跡する必要がないため、パラメータを指定しません。

**例 3**

```
[Financial]

tran=FIN01

tran=FIN02 range=0.1,0.5,1,2,3,4,5,10,20 slo=1.0

tran=FIN03 range=0.1,0.5,1,2,3,4,5,10,20 slo=2.0
```

Financial アプリケーションのトランザクションです。FIN02 および FIN03 は、それぞれ固有のパラメータを使用します。FIN01 トランザクションは、時間区分およびサービスレベルの目標を追跡する必要がないため、パラメータを指定しません。

**例 4**

```
[PERSONL]
```

```
tran=PERS* range=0.1,0.5,1,2,3,4,5,10,20 slo=1.0

tran=PERS03 range=0.1,0.2,0.5,1,2,3,4,5,10,20 slo=0.8
```

PERS03 アプリケーションのトランザクションです。PERSONL は固有のパラメータを使用します。残りの人事アプリケーショントランザクションは、PERSONL アプリケーションに固有のデフォルトのパラメータセットを使用します。

**例 5**

```
[ACCOUNTS]

tran=ACCT_* slo=1.0

tran=ACCT_REC range=0.5,1,2,3,4,5,10,20 slo=2.0

tran=ACCT_PAY range=0.5,1,2,3,4,5,10,20 slo=2.0
```

ACCOUNTS アプリケーションのトランザクションです。ACCT_REC および ACCT_PAY トランザクションは、それぞれ固有のパラメータを使用します。残りの会計トランザクションは、会計アプリケーションに固有のデフォルトのパラメータセットを使用します。買掛金および売掛金トランザクションだけが時間区分を追跡する必要があります。トランザクション名の順序によって、アプリケーションで違いが生じることはありません。

# ARM を使用する際のオーバーヘッドの考慮事項

現在のバージョンの Performance Collection Component および GlancePlus には、ARM 2.0 に必要な追加のデータをサポートする測定インターフェイスの修正が含まれています。これらの修正により、パフォーマンス管理上のオーバーヘッドの増加が発生します。アプリケーションに ARM を組み込む場合、オーバーヘッドについて考慮してください。

オーバーヘッドについては、この章の後で説明されています。

## ガイドライン

アプリケーションに ARM API を組み込むためのガイドラインを次に示します。

- 個別のトランザクション ID の数の合計は最高 4,000 に制限されます。一般に、異なるトランザクションの単一インスタンスを持つよりも、同じトランザクションの複数のインスタンスを持つより方が安価です。実際に監視されるトランザクションのみを登録してください。
- arm_start および arm_stop の API コールによるオーバーヘッドは非常に小さいものですが、多くのトランザクション インスタンスがある場合、オーバーヘッドが増加することがあります。ほとんどのシステム上で、1 秒単位あたりに数千を超える arm_start および arm_stop の API コールは、全体的なパフォーマンスに著しく影響を与えます。
- ARM 2.0 の機能を使用する場合にのみ ARM 相関係数を使用してください (ARM 相関係数の詳細については、『*アプリケーション応答測定 2.0 API ガイド*』の「高度なトピック」のセクションを参照してください)。相関係数情報の作成、移動、監視などのオーバーヘッドは、ARM 2.0 相関係数機能を使用しないトランザクションの監視よりも著しく高くなります。
- より大きな文字列サイズ (長いトランザクション名、アプリケーション名およびユーザー定義の文字列メトリックを登録するアプリケーション) は、さらにオーバーヘッドを増加させます。

# ディスク I/O オーバーヘッド

パフォーマンス管理ソフトウェアは、システム上で大きなディスクオーバーヘッドにはなりません。通常 Glance は、ディスクにそのデータを記録しません。Performance Collection Component の収集デーモン (scope) はディスクログファイルを生成しますが、ARM 2.0 によってそれらのサイズが影響を受けることはほとんどありません。logtran scope ログファイルは ARM データを格納するために使用されます。

# CPU オーバーヘッド

ARM コールを組み込んだプログラムは通常、ARM コールによる速度の低下はありません。これは、arm_getid コールの割合が毎秒 1 回未満のコールであり、また、arm_start および arm_stop コールの割合は毎秒数千より少ないと仮定した場合です。これ以上に頻繁な ARM API へのコールは避けてください。

ARM をサポートために発生する追加の CPU オーバーヘッドの多くは、パフォーマンスツールプログラムおよびデーモンの内部で発生します。midaemon CPU のオーバーヘッドはわずかに増加しますが、ARM 1.0 に比較して 2 パーセント以下です。midaemon がトランザクションごとのリソースメトリックを追跡するように要求されている場合、トランザクションインスタンスごとのオーバーヘッドは、トランザクションごとのリソースメトリックを追跡しない場合の 2 倍程度です(parm ファイル内で log transaction=resource フラグを設定することにより、トランザクションごとのリソースメトリックの追跡が可能です)。さらに、Glance および scope のCPU オーバーヘッドは、ARM 2.0 コールが組み込まれたアプリケーションのあるシステムでわずかに高くなります。相関係数やユーザー定義メトリックを拡張利用する、ARM 2.0 コールが組み込まれたアプリケーションだけが、midaemon、scope または Glance 上でのパフォーマンスに大きく影響します。

利用可能なデフォルトの共有メモリーを使用率が超過した場合、midaemon オーバーフロー状態が発生することがあります。この結果、次を引き起こします。

- オーバーフローの状態が発生すると、ARM コールからのリターンコードがない。
- 空白のプロセス名など、不正なメトリックが表示される。
- 「out of space」などのエラーが status.mi に記録される。

# メモリー オーバーヘッド

ARM API コールを生成するプログラムは、ARM 2.0 相関係数およびユーザー定義メトリック情報を渡すために使用されるスペースを除いて、それらのメモリー仮想セット サイズに大きな影響はありません。『*アプリケーション応答測定 2.0 API ガイド*』で説明されているこれらのバッファは、プロセスのメモリーの必要条件に影響を与えません。

ARM 2.0 をサポートするパフォーマンスツールでは、追加の仮想セット サイズのオーバーヘッドがあります。Performance Collection Component および Glance によって使用されるために ARM データが内部に保持される場合、midaemon プロセスは共有メモリー セグメントを作成します。この共有メモリー セグメントのサイズは、ARM 2.0 により使用される可能性を含めたために、ARM 1.0 リリースでのサイズに比べて大きくなりました。ほとんどのシステム上のデフォルトでは、この共有メモリー セグメントのサイズは、およそ 11 メガバイトです。必要でない場合、このセグメントが物理メモリーに必ずあるとは限りません。したがって、まだメモリーの制約を受けていないほとんどのシステムには大きな影響を与えません。midaemon のメモリー オーバーヘッドは、特別な開始パラメータを使用して調整できます (*midaemon* のマニュアル ページを参照してください)。

# トランザクションについて

この章では、トランザクション追跡とサービスレベルの目標管理の開始に必要な情報を提供します。詳細は、Chapter 19, How Transaction Tracking Works を参照してください。また、使用例については、Chapter 23, Transaction Tracking Examples を参照してください。

# 開始する前に

Performance Collection Component は、`libarm.*` 共有ライブラリを次の場所に配置します。

| プラットフォーム | パス |
|---|---|
| IBM RS/6000 | `/usr/lpp/perf/lib/` |
| その他の UNIX プラットフォーム | `/opt/perf/lib/` |

Performance Collection Component がシステムにインストールされていない場合、または `libarm.*` が上記のプラットフォームのパスに存在しない場合、このマニュアルの最後のC Compiler Option Examples by Platform on page 378を参照してください。また、入手については『アプリケーション応答測定 2.0 API ガイド』の「ARM 共有ライブラリ (`libarm`)」を参照してください。`libarm` については、このマニュアルの最後のARM Library (libarm) on page 373を参照してください。

# トランザクション追跡のセットアップ

アプリケーションにトランザクション追跡を設定するには、次の手順を行います。これらの手順については、この章の後で詳細を説明します。

1. 監視する主要なトランザクションおよび期待する応答レベルを決定し、SLO を定義します (オプション)。
2. Performance Collection Component および Performance Manager のトランザクションを監視する場合、Performance Collection Component の `parm` ファイルのトランザクション ログがオンになっていることを確認してください。その後、Performance Collection Component を開始または再起動して、最新の `parm` ファイルを読み込みます。

   GlancePlus でトランザクションを見るために `parm` ファイルを編集する必要はありません。ただし、GlancePlus でトランザクション データを見るには、ttd が実行されている必要があります。GlancePlus を開始すると、自動的に `ttd` が起動します。
3. このマニュアルと『*アプリケーション応答測定 2.0 API ガイド*』で説明されている ARM API コールが組み込まれたアプリケーションを実行します。
4. 収集されたトランザクション データを見るには、Performance Collection Component または Performance Manager を使用します。または、現在のデータを見るには GlancePlus を使用します。Performance Manager でデータが表示されない場合、データ ソースを閉じてから再度接続します。
5. アプリケーションのトランザクション データを収集する方法を修正するには、構成ファイル `ttd.conf` をカスタマイズします (オプション)。

6. ttd.conf ファイルに追加を行ったら、次の手順を実行して、追加を有効にする必要があります。
   a. すべての ARM 組み込みアプリケーションを終了する。
   b. ttd -hup -mi コマンドを root として実行する。

   これらの操作を実行すると、ttd.conf ファイルが再度読み込まれ、新しいトランザクションが slo と range の値と共に ttd および midaemon に登録されます。再度読み込んでも、再読み込み前に ttd.conf ファイルにあったトランザクションの slo と range の値は変更されません。
7. ttd.conf ファイルで、既存のトランザクションの slo と range の値を変更するには、次を実行します。
   a. すべての ARM 組み込みアプリケーションを終了する。
   b. ovpa stop を使用して scope コレクタを終了する。
   c. Glance の使用をすべて停止する。
   d. ttd -k コマンドを使用して ttd を終了する。

   変更を加えたら、
   a. ovpa start を使用して、scope を再起動する。
   b. ARM を含むアプリケーションを開始する。

## サービス レベルの目標 (SLO) の定義

トランザクション追跡を実装する最初のステップは、顧客の期待に応えるために必要となる主要なトランザクションと、トランザクションの応答レベル値を決定することです。要求される応答レベル値がサービス レベルの目標 (SLO) になります。サービス レベルの目標は、トランザクション構成ファイル ttd.conf に定義します。

サービス レベルの目標 (SLO) の定義は、IT 部門のサービス レベル協定 (SLA) を見直して、サービス レベル協定 (SLA) を満たすために監視する必要のあるトランザクションを確認する程度の簡単なものでもかまいません。SLA がなければ、導入されることをお勧めします。ただし、SLA の作成はトランザクションを追跡するために必須ではありません。

## parm ファイルの変更

必要に応じて、Performance Collection Component の parm ファイルを変更し、Performance Manager および Performance Collection Component で使用するために、記録する項目のリストにトランザクションを追加します。トランザクションを追加するには、次の例のように parm ファイルの log パラメータに transaction オプションを追加します。

```
log global application process transaction device=disk
```

log transaction パラメータのデフォルトは、no resource と no correlator です。リソース データ収集または相関係数データの収集を行うには、log transaction=resource または log transaction=correlator を指定します。両方とも、log transaction=resource、correlator を指定することで記録できます。

Performance Collection Component および Performance Manager で使用するトランザクション データを収集する前に、次の方法で最新の parm ファイルを起動させる必要があります。

| Performance Collection Component の状態 | トランザクション追跡の起動コマンド |
|---|---|
| 実行中 | ovpa restart |
| 停止中 | ovpa start |

# トランザクション データの収集

アプリケーションを開始します。トランザクション追跡デーモン (ttd)、および測定インターフェイス デーモン (midaemon) は、アプリケーションの実行に同期してトランザクション データを収集します。データは、Performance Collection Component または GlancePlus が使用できるように、midaemon の共有メモリー セグメントに格納されます。アプリケーションの各トランザクション データを表示する各ツールについては、「パフォーマンス データの監視 」(340ページ)を参照してください。

## エラーの対応

パフォーマンスの考慮事項にあるように、問題のあるすべての ARM API コールまたは TT API コールが、リアルタイムにエラーを返すとは限りません。エラーが期待どおりに返らない場合の例を次に示します。

- 初期化されていない変数など、問題のある id パラメータを指定して arm_start をコールした場合。
- 前回有効な arm_start コールを行わずに、arm_stop をコールした場合。

Performance Collection Component - アプリケーションに ARM コールを組み込むときにこれらの状況をデバッグするには、十分な量のトランザクション データを生成および収集できるだけの時間、アプリケーションを実行します。Performance Collection Component でデータを収集した後に、extract プログラムの export コマンドを使用して、logtran ファイルからデータをエクスポートします。データを検査して、必要なトランザクションがすべて記録されていることを確認します。また、エラーに関しては、/var/opt/perf/status.ttd ファイルを参照してください。

GlancePlus - アプリケーションに ARM コールを組み込むときにこれらの状況をデバッグするには、十分な量のトランザクション データを生成および収集できるだけの時間、アプリケーションを実行します。次に、GlancePlus を使用して、すべてのトランザクションが期待どおりに表示されることを確認します。

## 固有のトランザクションの制限

特定のシステム リソースおよびカーネルの設定によっては、アプリケーションで許容される固有のトランザクションの数が制限される場合があります。この上限は通常、数千回の固有の arm_getid コールです。

midaemon によって使用される共有メモリー セグメントがいっぱいになると、固有のトランザクションの数が制限を超える場合があります。この場合、GlancePlus にオーバーフロー メッセージが表示されます。Performance Collection Component ではこのメッセージは表示されず、以後の新しいトランザクション データは記録されません(ただし、オーバーフロー メッセージは /var/opt/perf/status.scope で確認できます)。GlancePlus では、以後の新しいトランザクション データは表示されません。すでに登録されているトランザクションは引き続き記録および報告されます。測定できなかった新しいトランザクションの数は、GlancePlus の GBL_TT_OVERFLOW_COUNT メトリックで報告されます。

この状態は、-smdvss オプションを使用してより大きな共有メモリー セグメントのサイズを指定し、midaemon プロセスを終了して再起動することにより改善できます。現在の共有メモリー セグメントのサイズは、midaemon -sizes のコマンドを使用して確認できます。システムでの midaemon の最適化の詳細は、*midaemon* のマニュアル ページを参照してください。

# 構成ファイルのカスタマイズ (オプション)

アプリケーションからのトランザクション データを確認した後、アプリケーションのトランザクション データの収集方法を変更する場合、トランザクション構成ファイル /var/opt/perf/ttd.conf をカスタマイズできます。デフォルトの構成ファイル ttd.conf は、アプリケーションで定義されたすべてのトランザクションを取り扱うため、変更はオプションです。ttd.conf ファイルをカスタマイズする場合は、アプリケーションを実行するシステム上でこの作業を行ってください。ttd.conf を変更するには、root としてログオンする必要があります。

構成ファイル キーワード tran、range、slo についての情報は、Chapter 19, How Transaction Tracking Worksを参照してください。各キーワードの使用例を次に示します。

```
tran=Example:    tran=answerid

                 tran=answerid*

                 tran=*

range=Example:   range=2.5,4.2,5.0,10.009

slo=Example:      slo=4.2
```

トランザクションとそれに関連する属性をすべて含むように、構成ファイルを編集します。tran キーワードは、range キーワードまたは slo キーワードの前に使用する必要があります。ttd.conf 構成ファイルの例を次に示します。

```
 tran=*

 tran=my_first_transaction slo=5.5

 [answerid]

 tran=answerid1 range=2.5, 4.2, 5.0, 10.009 slo=4.2

 [orderid]

 tran=orderid1 range=1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0
```

ttd.conf ファイルに追加が必要な場合、次のように行ってください。

- すべての ARM 組み込みアプリケーションを終了する。
- ttd -hup -mi コマンドを root として実行する。

上記の操作を実行すると、ttd.conf ファイルが再度読み込まれ、新しいトランザクションが slo と range の値と共に ttd および midaemon に登録されます。再度読み込んでも、再読み込み前に ttd.conf ファイルにあったトランザクションの slo と range の値は変更されません。

ttd.conf ファイルで、既存のトランザクションの slo と range の値を変更するには、次を実行します。

1. すべての ARM 組み込みアプリケーションを終了する。
2. ovpa stop を使用して scope コレクタを終了する。

3. Glance の使用をすべて停止する。
4. `ttd -k` コマンドを使用して `ttd` を終了する。

変更を加えたら、

1. `ovpa start` を使用して、`scope` を再起動する。
2. ARM を含むアプリケーションを開始する。

# パフォーマンス データの監視

次のリソースと Performance Collection Component、Performance Manager、GlancePlus などのパフォーマンス管理製品を使用して、トランザクション データを監視できます。

**Performance Collection Component での使用 Performance Collection Component**

Performance Collection Component で、長期間のデータ収集および記録を行うことにより、システム パフォーマンスの分析および傾向分析が可能となります。Performance Collection Component のデータは、Performance Manager エージェントで表示できる他、さまざまな他のパフォーマンス監視、会計報告、モデリング、計画ツールなどに使用するためにエクスポートできます。

Performance Collection Component の `extract` プログラムにより、データはスプレッドシート、および分析プログラムに使用するようにエクスポートできます。データはアーカイブと分析のために抽出できます。

Performance Collection Component でトランザクション データを監視するには、Performance Collection Component および ttd が実行されている必要があります。ovpa スクリプトを使用して Performance Collection Component を開始することで、GlancePlus によるトランザクション データの監視に必要な `ttd` および `midaemon` プロセスが、正しい順序で開始されます。

**Performance Manager での使用**

Performance Manager は Performance Collection Component のデータをインポートし、カスタマイズされたグラフ形式やスプレッド シート形式に変換して表示します。Performance Manager により、トランザクション データの履歴傾向の分析、将来予測をより正確に実行できます。

Performance Manager のデータ ソースとして、[クラス リスト] ウィンドウから **TRANSACTION** を選び、さまざまなトランザクション メトリックをグラフ化できます。詳細は、Performance Manager のヘルプ メニューからアクセスできる Performance Manager のオンライン ヘルプを参照してください。Performance Manager で目的のトランザクションが表示されない場合、現在のデータ ソースを閉じてから再度接続してください。

**GlancePlus での使用**

GlancePlus を使用してシステムを監視することにより、リソースのボトルネックの識別や、コンピュータ システムに関するパフォーマンス情報をすぐにで得られます。GlancePlus には、定義したすべてのトランザクションに関する情報を表示する [トランザクション追跡] ウィンドウ、および個別のトランザクションに関する特定の情報を表示する [トランザクション グラフ] ウィンドウがあります。たとえば、定義した SLO に対する各トランザクションの実行状況がわかります。GlancePlus の使用方法の詳細は、ヘルプ メニューからアクセスできるオンライン ヘルプを参照してください。

# アラーム

Performance Collection Component、Performance Manager、GlancePlus のリソースおよびパフォーマンス管理製品により、トランザクション データに関するアラームを生成することができます。

**... Performance Collection Component での使用**

Performance Collection Component でアラームを生成するには、アラーム定義ファイル alarmdef でアラーム状態を定義する必要があります。E-mail メッセージの送信やページャの呼び出しなど、さまざまな方法でアラーム状態を通知するように Performance Collection Component を設定できます。

alarmdef ファイルの構文チェックをパスさせるには、ログファイルにそのアプリケーション名とトランザクション名のデータを記録するか、または ttd.conf ファイルに名前の登録をしておく必要があります。

名称にダッシュ (-) を含むトランザクションにアラーム状態を定義する場合には、制限があります。この制限を回避するには、alarmdef ファイル内で ALIAS コマンドを使用してトランザクション名を再定義します。

**GlancePlus での使用**

トランザクション パフォーマンスに関するアラームを生成するようにアドバイザ シンタックスを設定することができます。たとえば、アラーム状態が満たされた場合、情報を stdout に表示したり、UNIX コマンド (mailx など) を実行したり、GlancePlus メイン ウィンドウ上の [アラーム] ボタンを黄色や赤色に変えたりするように、GlancePlus に指示できます。GlancePlus のアラームの詳細は、[アドバイザ シンタックスの編集] ウィンドウのヘルプ メニューから [**ウィンドウについて**] を選択してください。

# トランザクション追跡のメッセージ

アプリケーションに ARM API コールまたは TT API コールを組み込む場合、表 2 に示すエラー コードが返されるため、アプリケーション開発者はそれらを利用することができます。

**表 2:エラー コード**

| エラー コード | Errno 値 | 説明 |
|---|---|---|
| -1 | EINVAL | 無効な引数 |
| -2 | EPIPE | ttd (登録デーモン) が実行されていない |
| -3 | ESRCH | ttd.conf ファイルにトランザクション名がない |
| -4 | EOPNOTSUPP | オペレーティング システムのバージョンがサポートされていない |

ARM API コールまたは TT API コールが組み込まれたアプリケーションが実行されている場合、発生するエラーのリターン コードはトランザクション追跡デーモン ttd から返されます。測定インターフェイス デーモン midaemon はいかなるエラー リターン コードも生成しません。

midaemon エラーが発生する場合、内容の詳細は /var/opt/perf/status.mi ファイルを参照してください。

# トランザクション メトリック

ARM エージェントは、GlancePlus とPerformance Collection Component の共有コンポーネントとして提供され、さまざまなトランザクション メトリックを作成します。次の方法で、メトリックとその説明についての詳細なリストを表示できます。

- インストールされている GlancePlus のメトリックについては、GlancePlus のオンライン ヘルプを使用するか、または次の場所にある『*GlancePlus for HP-UX Dictionary of Performance Metrics*』を参照してください。

  UNIX/Linux 上の `/<InstallDir>/paperdocs/gp/C/` にある `gp-metrics.txt`。

  `InstallDir` は、Performance Collection Component がインストールされているディレクトリです。

- 特定のプラットフォームにインストールされている Performance Collection Component のメトリックについては、次の場所にある、プラットフォームに対応した「*HP Operations Agent Dictionary of Operating System Performance Metrics*」ファイルを参照してください。

  UNIX/Linux 上の `/<InstallDir>/paperdocs/ovpa/C/` にある `met<`*プラットフォーム*`>.txt`。

  Windows 上の *%ovinstalldir%*`paperdocs\ovpa\C` にある `met<`*プラットフォーム*`>.txt`。

# トランザクション追跡の例

この章には、アプリケーションにARM API コールを組み込み、アプリケーションに定義されたトランザクションを Performance Collection Component または GlancePlus で監視できるようにする擬似コード例が含まれています。この擬似コード例は、Chapter 18, What is Transaction Tracking?で説明されているリアルタイム注文処理のシナリオに対応します。

この章には、リアルタイム注文処理シナリオに対応する構成ファイルの例など、いくつかのトランザクション構成ファイルの例が含まれています。

## リアルタイム注文処理の擬似コード

この擬似コード例には、
Chapter 18, What is Transaction Tracking?で説明されている、リアルタイム注文処理シナリオのトランザクションを定義するために使用された ARM API コールが含まれています。オペレータが電話注文を受けるたびに、このルーチンが処理されます。この例では、ARM API コールを含む行は太字で示されます。

```
routine answer calls()

{

*****************************************************

*  Register the transactions if first time in       *

*****************************************************

    if (transactions not registered)

    {

          appl_id = arm_init("Order Processing Application","*",
0,0,0)

          answer_phone_id = arm_getid(appl_id,"answer_phone","1st
tran",0,0,0)

          if (answer_phone_id < 0)

              REGISTER OF ANSWER_PHONE FAILED - TAKE APPROPRIATE
ACTION

          order_id = arm_getid(appl_id,"order","2nd tran",0,0,0)

          if (order_id < 0)

              REGISTER OF ORDER FAILED - TAKE APPROPRIATE ACTION

          check_id = arm_getid(appl_id,"check_db","3rd tran",0,0,0)


          if (check_id < 0)

             REGISTER OF CHECK DB FAILED - TAKE APPROPRIATE ACTION
```

```
          update_id = arm_getid(appl_id,"update","4th tran",0,0,0)

          if (update_id < 0)
             REGISTER OF UPDATE FAILED - TAKE APPROPRIATE ACTION
     } if transactions not registered
*****************************************************
*  Main transaction processing loop
*****************************************************
        while (answering calls)
        {
             if (answer_phone_handle = arm_start(answer_phone_id,0,0,0) < -1)

            TRANSACTION START FOR ANSWER_PHONE NOT REGISTERED
*****************************************************
*  At this point the answer_phone transaction has    *
*  started.  If the customer does not want to order, *
*  end the call; otherwise, proceed with order.    *
*****************************************************
        if (don't want to order)
            arm_stop(answer_phone_handle,ARM_FAILED,0,0,0)
            GOOD-BYE - call complete
        else
         {
*****************************************************
* They want to place an order - start an order now  *
*****************************************************
          if (order_handle = arm_start(order_id,0,0,0) < -1)

            TRANSACTION START FOR ORDER FAILED
          take order information: name, address, item, etc.
*****************************************************
* Order is complete - end the order transaction    *
*****************************************************
        if (arm_stop(order_handle,ARM_GOOD,0,0,0) < -1)
                TRANSACTION END FOR ORDER FAILED
```

```
*****************************************************
*  order taken - query database for availability     *
*****************************************************
        if (query_handle = arm_start(query_id,0,0,0) < -1)
                TRANSACTION QUERY DB FOR ORDER NOT REGISTERED
        query the database for availability
****************************************************
* database query complete - end query transaction  *
****************************************************
        if (arm_stop(query_handle,ARM_GOOD,0,0,0) < -1)
                TRANSACTION END FOR QUERY DB FAILED
*****************************************************
* If the item is in stock, process order, and        *
* update inventory.                                  *
*****************************************************
        if (item in stock)
        if (update_handle = arm_start(update_id,0,0,0) < -1)
                TRANSACTION START FOR UPDATE NOT REGISTERED
        update stock
****************************************************
* update complete - end the update transaction     *
****************************************************
        if (arm_stop(update_handle,ARM_GOOD,0,0,0) < -1)
                TRANSACTION END FOR ORDER FAILED
*****************************************************
* Order complete - end the call transaction          *
*****************************************************
        if (arm_stop(answer_phone_handle,ARM_GOOD,0,0,0) < -1)
                TRANSACTION END FOR ANSWER_PHONE FAILED
        } placing the order
        GOOD-BYE - call complete
        sleep("waiting for next phone call...zzz...")
        }       while answering calls
```

```
        arm_end(appl_id, 0,0,0)
        }       routine answer calls
```

# 構成ファイル例

このセクションには、いくつかのトランザクション構成ファイル /var/opt/perf/ttd.conf の例が含まれています。ttd.conf ファイルおよび構成ファイルのキーワードの詳細は、Chapter 19, How Transaction Tracking Worksを参照してください。

## 例 1 (注文処理の擬似コード例)

```
# The "*" entry below is used as the default if none of the
# entries match a registered transaction name.

tran=* range=0.5,1,1.5,2,3,4,5,6,7 slo=1
tran=answer_phone* range=0.5,1,1.5,2,3,4,5,6,7 slo=5
tran=order* range=0.5,1,1.5,2,3,4,5,6,7 slo=5
tran=query_db* range=0.5,1,1.5,2,3,4,5,6,7 slo=5
```

## 例 2

```
# The "*" entry below is used as the default if none of the
# entries match a registered transaction name.

tran=* range=1,2,3,4,5,6,7,8 slo=5

# The entry below is for the only transaction being
# tracked in this application.The "*" has been inserted
# at the end of the tran name to catch any possible numbered
# transactions.For example "First_Transaction1",
# "First_Transaction2", etc.

tran=First_Transaction* range=1,2.2,3.3,4.0,5.5,10 slo=5.5
```

## 例 3

```
# The "*" entry below is used as the default if none of the
# entries match a registered transaction name.
```

```
tran=*
tran=Transaction_One range=1,10,20,30,40,50,60 slo=30
```

## 例 4

```
tran=FactoryStor* range=0.05, 0.10, 0.15 slo=3
# The entries below shows the use of an application name.
# Transactions are grouped under the application name.This
# example also shows the use of less than 10 ranges and
# optional use of "slo."

[Inventory]
tran=In_Stock range=0.001, 0.004, 0.008
tran=Out_Stock range=0.001, 0.005
tran=Returns range=0.1, 0.3, 0.7

[Pers]
tran=Acctg range=0.5, 0.10, slo=5
tran=Time_Cards range=0.010, 0.020
```

# 高度な機能

この章では、次の ARM 2.0 API 機能を Performance Collection Component で使用する方法を説明します。

- データ型
- ユーザー定義メトリック
- scope の組み込み

## データ型の使用方法

下の表は、Performance Collection Component でのデータ型の使用方法を示します。これは『*アプリケーション応答測定 2.0 API ガイド*』の「高度なトピック」セクションの「データ型の定義」の補足です。

**表 3:Performance Collection Component でのデータ型の使用方法**

| | |
|---|---|
| ARM_Counter32 | データは 32 ビットの整数で記録されます。 |
| ARM_Counter64 | データは 32 ビットの整数で、型変換されて記録されます。 |
| ARM_CntrDivr32 | 計算された結果が 32 ビットの整数として記録されます。 |
| ARM_Gauge32 | データは 32 ビットの整数で記録されます。 |
| ARM_Gauge64 | データは 32 ビットの整数で、型変換されて記録されます。 |
| ARM_GaugeDivr32 | 計算された結果が 32 ビットの整数として記録されます。 |
| ARM_NumericID32 | データは 32 ビットの整数で記録されます。 |
| ARM_NumericID64 | データは 32 ビット整数で、型変換されて記録されます。 |
| ARM_String8 | 無視されます。 |
| ARM_String32 | 無視されます。 |

Performance Collection Component は文字列データを記録しません。Performance Collection Component は 5 分ごとにデータを記録し、その間隔でのアクティビティの要約を記録するためです。アプリケーションから提供される文字列は要約できません。

Performance Collection Component は使用可能な最初の 6 つのユーザー定義メトリックから、その最大、最小、平均を記録します。ARM が組み込まれたアプリケーションは、Counter32、String8、NumericID32、Gauge32、Gauge64、Counter64、NumericID64、String32、GaugeDivr32 を渡し、Performance Collection Component は 32 ビットの整数として、Counter32、NumericID32、Gauge32、Gauge64、NumericID32、NumericID64 の Min、Max、Average を 5 分間隔で記録します。Performance Collection Component では文字列を要約できないため、String8 と String32 は無視されます。最初の 6 つのユーザー定義メトリックのみが記録されるため、GaugeDivr32 も無視されます(その他の例は、次のセクションの「ユーザー定義メトリック」を参照してください)。

# ユーザー定義メトリック

このセクションは『 *アプリケーション応答測定 2.0 API ガイド*』の「高度なトピック」の「アプリケーション定義メトリック」の補足です。ここでは、Performance Collection Component がユーザー定義メトリック (ARM ではアプリケーション定義メトリックと呼ばれているもの) を処理する方法の例を示します。表 4 の例では、プログラムが次のようなデータ型を渡した場合に記録される内容を示します。

**表 4:記録される特定のプログラムのデータ型の例**

| プログラムが渡す内容 | 記録される内容 |
|---|---|
| **例 1** | |
| `String8` | |
| `Counter32` | `Counter32` |
| `Gauge32` | `Gauge32` |
| `CntrDivr32` | `CntrDivr32` |
| **例 2** | |
| `String32` | |
| `NumericID32` | `NumericID32` |
| `NumericID64` | `NumericID64` |
| **例 3** | |
| `NumericID32` | `NumericID32` |
| `String8` | |
| `NumericID64` | `NumericID64` |
| `Gauge32` | `Gauge32` |
| `String32` | |
| `Gauge64` | `Gauge64` |
| **例 4** | |
| `String8` | (なし) |
| `String32` | |
| **例 5** | |
| `Counter32` | `Counter32` |
| `Counter64` | `Counter64` |
| `CntrDivr32` | `CntrDivr32` |
| `Gauge32` | `Gauge32` |

| | |
|---|---|
| Gauge64<br>NumericID32<br>NumericID64 | Gauge64<br>NumericID32 |

Performance Collection Component は文字列データを要約できないため、文字列データは記録されません。

例 1 では、カウンタ、ゲージ、カウンタ除数のみが記録されます。

例 2 では、数値のみが記録されます。

例 3 では、数値とゲージのみが記録されます。

例 4 では、何も記録されません。

例 5 では、最初の 6 つのユーザー定義メトリックのみが記録されるため、NumericID64 は記録されません。

# scope の組み込み

ARM API コールには、scope データコレクタが組み込まれています。Performance Collection Component が起動すると、scope は自動的に Scope_Get_Process_Metrics および Scope_Get_Global_Metrics の 2 つのトランザクションの記録を開始します。この 2 つのトランザクションは HP パフォーマンス ツールのアプリケーションにあります。

トランザクション データは 5 分ごとに記録されるため、各間隔で 5 つの Get Process トランザクション (1 分につき 1 つのトランザクション) が完了します。Scope_Get_Process_Metrics トランザクションはプロセス データの処理に組み込まれます。システムに 200 のプロセスがある場合、Scope_Get_Process_Metrics トランザクションには、システムに 30 のプロセスしかない場合より時間がかかります。

Scope_Get_Global_Metrics トランザクションは、グローバル データを含むすべての 5 分間隔のデータの収集に組み込まれます。これには、global、application、disk、transaction、およびその他のデータ型が含まれます。

プロセスとグローバルトランザクション データの記録を停止するには、ttd.conf ファイルにある scope トランザクションのエントリをコメント アウトするか削除します。

# トランザクション ライブラリ

この付 録 では、次 の項 目 について説 明します。

- アプリケーション応 答 測 定ライブラリ (`libarm`)
- プラットフォームごとの C コンパイラ オプションの例
- アプリケーション応 答 測 定 NOP ライブラリ (`libarmNOP`)
- Java ラッパーの使 用

## ARM ライブラリ (libarm)

Performance Collection Component および GlancePlus を使 用 すると、ARM の機 能 のコンパイルおよび使 用 を簡 単 に行 える環 境 を構 築 できます。開 発 に必 要 なライブラリは `/opt/perf/lib/` に配 置 されています。コンパイルに関 する特 定 の情 報 は、この付 録 の次 のセクションを参 照 してください。

表 5 に示 すライブラリ ファイルは HP-UX 11.11 以 降 の Performance Collection Component および GlancePlus のインストールに存 在 します。

**表 5:HP-UX 11.11 以 降 の Performance Collection Component および GlancePlus のライブラリ ファイル**

| | | |
|---|---|---|
| `/opt/perf/lib/` | `libarm.0` | ARM の HP-UX 10.X 互 換 共 有 ライブラリ (非 スレッド セーフ) です。`-larm` を指 定 して HP-UX 10.20 にリンクされている HP-UX 11 でプログラムを実 行 した場 合 、11.0 ローダーは自 動 的 にこのライブラリを参 照 します。 |
| | `libarm.1` | HP-UX 11 の互 換 共 有 ライブラリ (スレッド セーフ) です。このライブラリは、`-larm` を指 定 して HP-UX リリースにリンクされているプログラムにより参 照 されます。10.20 にリンクするプログラムがこのライブラリを参 照 する場 合 (たとえば、`-L /opt/perf/lib` にリンクしていない場 合 )、「`/usr/lib/dld.sl: Unresolved symbol: _thread_once (code) from libtt.sl`」というエラー メッセージが表 示 され、中 断 されます。 |
| | `libarm.sl` | `libarm.1` のシンボリック リンクです。 |
| | `libarmNOP.sl` | ARM の「非 操 作 用 」共 有 ライブラリです (API コールは成 功 しますが何 もしません。Performance Collection Component がインストールされていないシステム上 でテストする場 合 に使 用 します)。 |

| | | |
|---|---|---|
| /opt/perf/examples/arm | libarmjava.sl | ARM の 32 ビット 共 有 ライブラリです。 |
| /opt/perf/examples/arm/arm64 | libarmjava.sl | ARM の 64 ビット 共 有 ライブラリです。 |
| /opt/perf/lib/pa20_64/ | これらのファイルは、HP-UX 11 上 で +DD64 コンパイラ オプションでコンパイルされるプログラムにより自 動 的 に参 照 されます。 | |
| | libarm.sl | ARM の 64 ビット 共 有 ライブラリです。 |
| | libarmNOP.sl | ARM の 64 ビット「非 操 作 用 」共 有 ライブラリです (API コールは成 功 しますが何もしません。Performance Collection Component がインストールされていないシステム上 でテスト する場 合 に使 用 します)。 |

表 6 に示 す追 加 のライブラリ ファイルは IA64 HP-UX インストールにあります。

**表 6:HP-UX IA64 ライブラリ ファイル**

| | | |
|---|---|---|
| /opt/perf/lib/hpux32/ | libarm.so.1 | ARM の IA64/32 ビット 共 有 ライブラリ |
| /opt/perf/lib/hpux64/ | libarm.so.1 | ARM の IA64/64 ビット 共 有 ライブラリ |
| /opt/perf/examples/arm | libarmjava.so | ARM の 32 ビット 共 有 ライブラリです。 |
| /opt/perf/examples/arm/arm64 | libarmjava.so | ARM の 64 ビット 共 有 ライブラリです。 |

ARM ライブラリがコールする HP-UX のオペレーティング システムのバージョンは次 のバージョンに変 わることがあるため、プログラムは -larm を使 って共 有 ライブラリ バージョンにリンクする必 要 があります。ARM API コールが組 み込 まれているアプリケーションのコンパイル、および ARM ライブラリのアーカイブ バージョンへのリンク (-Wl、-a archive) はサポートされていません(詳 細 は、2 章 のTransaction Tracking Daemon (ttd) on page 345を参 照 してください)。

Performance Collection Component と GlancePlus がインストールされている AIX オペレーティング システムに存 在 するライブラリ ファイルは、次 のとおりです。

**表 7:AIX ライブラリ ファイル**

| | | |
|---|---|---|
| /usr/lpp/perf/lib/ | libarm.a | 32 ビット 共 有 ARM ライブラリです (スレッド セーフ)。このライブラリは -larm を使 用 してリンクされているプログラムによって参 照 されます。 |

| | | |
|---|---|---|
| /usr/lpp/perf/lib | libarmNOP.a | ARM の 32 ビット 共 有 ライブラリです。このライブラリは、Performance Agent または Performance Collection Component がインストールされていないシステム上 でのテストに使 用 されます。 |
| /usr/lpp/perf/lib64/ | libarm.a | 64 ビット 共 有 ARM ライブラリです (スレッド セーフ)。このライブラリは -larm を使 用してリンクされているプログラムによって参照されます。 |
| /usr/lpp/perf/lib64 | libarmNOP.a | ARM の 64 ビット 共 有 ライブラリです。このライブラリは、Performance Agent または Performance Collection Component がインストールされていないシステム上 でのテストに使 用 されます。 |
| /usr/lpp/perf/examples/arm | libarmjava.a | ARM の 32 ビット 共 有 ライブラリ |
| /usr/lpp/perf/examples/arm/arm64 | libarmjava.a | ARM の 64 ビット 共 有 ライブラリです。 |
| /usr/lpp/perf/lib/ | libarmns.a | 32 ビット アーカイブ ARM ライブラリです ( スレッド セーフ)。機 能 的 には、32 ビット の libarm.a と同じです。 |
| /usr/lpp/perf/lib64/ | libarmns.a | 64 ビット アーカイブ ARM ライブラリです ( スレッド セーフ)。機 能 的 には、64 ビット の libarm.a と同じです。 |

Performance Collection Component と GlancePlus がインストールされている Solaris オペレーティング システムに存 在 するライブラリ ファイルは、次 のとおりです。

**表 8:32 ビット プログラム用 Solaris ライブラリ ファイル**

| | | |
|---|---|---|
| /opt/perf/lib/ | libarm.so | 32 ビット 共 有 ARM ライブラリです (スレッド セーフ)。このライブラリは -larm を使 用してリンクされているプログラムによって参 照 されます。 |
| | libarmNOP.so | ARM の 32 ビット 共 有 ライブラリです。このライブラリは、Performance Collection Component がインストールされていないシステム上 でのテストに使 用 されます。 |

**表 9:Sparc 64 ビット プログラム用 Solaris ライブラリ ファイル**

| | | |
|---|---|---|
| /opt/perf/lib/sparc_64/ | libarm.so | 64 ビット 共 有 ARM ライブラリです (スレッド セーフ)。このライブラリは -larm を使 用してリンクされているプログラムによって参 照 されます。 |
| | libarmNOP.so | ARM の 64 ビット 共 有 ライブラリ このライブラリは、Performance Agent または Performance Collection Component がインストールされていないシステム上 でのテストに使 用 されます。 |
| /opt/perf/examples/arm | libarmjava.so | ARM の 32 ビット 共 有 ライブラリです。 |
| /opt/perf/examples/arm/arm64 | libarmjava.so | ARM の 64 ビット 共 有 ライブラリです。 |

**表 10:x86 64 ビット プログラム用 Solaris ライブラリ ファイル**

| | | |
|---|---|---|
| /opt/perf/lib/x86_64/ | libarm.so | 64 ビット 共 有 ARM ライブラリです (スレッド セーフ)。このライブラリは -larm を使 用してリンクされているプログラムによって参 照 されます。 |
| | libarmNOP.so | ARM の 64 ビット 共 有 ライブラリ このライブラリは、Performance Agent がインストールされていないシステム上 でのテストに使 用 されます。 |
| /opt/perf/examples/arm/arm64 | libarmjava.so | ARM の 32 ビット 共 有 ライブラリです。 |
| /opt/perf/examples/arm/arm64 | libarmjava.so | ARM の 64 ビット 共 有 ライブラリです。 |

**注 :** 32 ビット プログラム用 に提 供 されているパラメータの他 に、`-xarch=generic64` コマンド ライン パラメータを使 用 して 64 ビット プログラムをコンパイルする必 要 があります。

Performance Collection Component と GlancePlus がインストールされている Linux オペレーティング システムに存 在 するライブラリ ファイルは、次 のとおりです。

**表 11:Linux ライブラリ ファイル**

| | | |
|---|---|---|
| /opt/perf/lib/ | libarm.so | 32 ビット 共 有 ARM ライブラリです (スレッド セーフ)。このライブラリは -larm を使 用してリンクされているプログラムによって参 照 されます。 |
| | libarmNOP.so | ARM の 32 ビット 共 有 ライブラリです。このライブラリは、Performance Collection Component がインストールされていないシステム上 でのテストに使 用 されます。 |

| | | |
|---|---|---|
| /opt/perf/lib/ | libarm.so | 64 ビット 共 有 ARM ライブラリです (スレッド セーフ)。このライブラリは -larm を使 用してリンクされているプログラムによって参 照 されます。 |
| | libarmNOP.so | ARM の 64 ビット 共 有 ライブラリです。このライブラリは、Performance Collection Component がインストールされていないシステム上 でのテストに使 用 されます。 |
| /opt/perf/examples/arm | libarmjava.so | ARM の 32 ビット 共 有 ライブラリです。 |
| /opt/perf/examples/arm/arm64 | libarmjava.so | ARM の 64 ビット 共 有 ライブラリです。 |

**注 :** Linux 2.6 IA 64 ビットでは、32 ビットの libarm.so と libarmjava.so は実 装 されていません。

# プラット フォームごとの C コンパイラ オプションの例

arm.h インクルード ファイルは、/opt/perf/include/ にあります。このファイルは /usr/include/ からシンボリック リンク経 由 でアクセスできます。つまり、-I/opt/perf/include/ を使 用 する必 要 はありません (使 用 することもできます)。同じように、libarm は /opt/perf/lib/ に存 在 しますが、/usr/lib/ からリンクされます。ARM 組 み込みアプリケーションを作 成 する場 合 、常 に -L/opt/perf/lib/ を使 用 してください。

- Linux の場 合 :
  次 の例 は、ARM API を使 用 する C プログラムのコンパイル コマンドを示します。

  ```
  cc myfile.c -o myfile -I /opt/perf/include -L -Xlinker -rpath -Xlinker /opt/perf/lib
  ```

- Linux の 64 ビット プログラムの場 合 :

  ```
  cc -m64 myfile.c -o myfile -I /opt/perf/include -L -Xlinker -rpath -Xlinker /opt/perf/lib64
  ```

- HP-UX の場 合 :
  IA64 プラットフォーム上 の HP-UX リリース 11.2x の場 合 、ARM 組 み込 み 32 ビット IA プログラムのコンパイルでは -L パラメータを -L/opt/perf/libから-L/opt/perf/lib/hpux32 に変 更 し、ARM 組 み込 み 64ビット IA プログラムのコンパイルでは -L/opt/perf/lib/hpux64 に変 更 します。

  次 の例 は、ARM API を使 用 する C プログラムのコンパイル コマンドを示します。

  ```
  cc myfile.c -o myfile -I /opt/perf/include -L /opt/perf/lib -larm
  ```

- Sun Solaris の場 合 :
  次 の例 は、Sun Solaris 上 の Performance Collection Component および GlancePlus で使 用 できます。

  ```
  cc myfile.c -o myfile -I /opt/perf/include -L /opt/perf/lib -larm -lnsl
  ```

- Sun Solaris の 64 ビット Sparc プログラムの場合 :
  次の例は、Sun Solaris 上の Performance Collection Component および 64 ビット プログラムで使用できます。

  ```
  cc -xarch=generic64 myfile.c -o myfile -I /opt/perf/include -L
  /opt/perf/lib/sparc_64 -larm -lnsl
  ```

- Sun Solaris の 64 ビット x86 プログラムの場合 :
  次の例は、Sun Solaris 上の Performance Agent および 64 ビット プログラムで使用できます。

  ```
  cc -xarch=generic64 myfile.c -o myfile -I /opt/perf/include -L
  /opt/perf/lib/x86_64 -larm -lnsl
  ```

- IBM AIX の場合 :
  IBM AIX 上のファイル配置は、他のプラットフォームと異なる (`/opt/perf/` の代わりに `/usr/lpp/perf/` が使用される) ため、IBM AIX の例は他のプラットフォームの例とは異なります。

  ```
  cc myfile.c -o myfile -I /usr/lpp/perf/include -L /usr/lpp/perf/lib
  -larm
  ```

- IBM AIX の 64 ビット プログラムの場合 :
  次の例は、IBM AIX 上の Performance Agent および 64 ビット プログラムで使用できます。

  ```
  cc -q64 myfile.c -o myfile -I /usr/lpp/perf/include -L
  /usr/lpp/perf/lib64 -larm
  ```

**注 :** C++ コンパイラでは、`arm.h` ファイルの適切な宣言を参照するために、コンパイル コマンドに `-D_PROTOTYPES` フラグを追加しなければならないことがあります。

# ARM NOP ライブラリ

Performance Collection Component と Glance には、「非操作用」ライブラリ (`libarmNOP.*` :「`*`」は、OS プラットフォームに応じて `sl`、`so`、`a` のいずれか) が提供されています。この共有ライブラリは、各 ARM API コールに対して有効な状態を返す以外のことは行いません。これにより、Performance Collection Component および GlancePlus がインストールされていないシステム上で、ARM が組み込まれているアプリケーションを動作できます。

Performance Collection Component または GlancePlus がインストールされていないシステム上で ARM が組み込まれているアプリケーションを実行するには、NOP ライブラリを適切なディレクトリ (通常は `/<InstallDir>/lib/`) にコピーして、`libarm.sl` (プラットフォームに応じて `libarm.so` または `libarm.a`) と名前を付けます。Performance Collection Component または GlancePlus がインストールされると、この NOP ライブラリは、正しく機能するライブラリによって上書きされます (他のファイルのように削除されません) 。これにより、システムから Performance Collection Component または GlancePlus が削除されても、組み込まれたプログラムは停止しません。

# Java ラッパーの使用

Java ネイティブ インターフェイス (JNI) ラッパーは、Java アプリケーションが HP ARM2.0 API をコールできるように作成された関数です。これらのラッパー (`armapi.jar`) は、ARM サンプル プログラムに含まれ、`/<InstallDir>/examples/arm/` ディレクトリに配置されています。`InstallDir` は、Performance Collection Component がインストールされているディレクトリです。

# 例

Java ラッパーの例 は、/<InstallDir>/examples/arm/ ディレクトリに配 置されています。また、この場 所には各 ラッパーの機 能を説 明する README ファイルも保 存されています。

## アプリケーションの設 定 (arm_init)

新 規 アプリケーションを設 定 するには、新しい ARMApplication のインスタンスを作 成し、この API の名 前 と説 明を渡します。各 アプリケーションは、固 有 の名 前 によって識 別 される必 要 があります。ARMApplication クラスは C の関 数 arm_init を使 用します。

### 構 文 :

```
ARMApplication myApplication =new ARMApplication("name","description")
```

## トランザクションの設 定 (arm_getid)

新しいトランザクションを設 定 するには、ユーザー定 義 メトリック (UDM) を使 用 するかどうかを選 択 できます。Java ラッパーは、C の関 数 arm_getid を使 用します。

## UDM を使 用したトランザクションの設 定

UDM を使 用 する場 合 は、まず新しい ARMTranDescription を定 義 する必 要 があります。ARMTranDescription は、arm_getid のデータ バッファを作 成します(jprimeudm.java の例 を参 照 )。

**構 文 :**

```
ARMTranDescription myDescription =
new ARMTranDescription("transactionName","details");
```

詳 細を使 用しない場 合 は、別 のコンストラクタを使 用 できます。

**構 文 :**

```
ARMTranDescription myDescription =
new ARMTranDescription("transactionName");
```

## メトリックの追 加

**メトリック 1-6:**

**構 文 :**

```
myDescription.addMetric(metricPosition, metricType,
metricDescription);
```

**パラメータ:**

```
metricPosition: 1-6
```

```
metricType: ARMConstants.ARM_Counter32
ARMConstants.ARM_Counter64 ARMConstants.ARM_CntrDivr32
ARMConstants.ARM_Gauge32 ARMConstants.ARM_Gauge64
ARMConstants.ARM_GaugeDivr32 ARMConstants.ARM_NumericID32
ARMConstants.ARM_NumericID64 ARMConstants.ARM_String8
```

**メトリック 7:**

**構文:**

```
myDescription.addStringMetric("description");
```

これで、トランザクションを作成できます。

**構文:**

```
myApplication.createTransaction(myDescription);
```

## メトリック データの設定

**メトリック 1-6:**

**構文:**

```
myTransaction.setMetricData(metricPosition, metric);
```

**「メトリック」の例**

```
ARMGauge32Metric metric = new ARMGauge32Metric(start);
ARMCounter32Metric metric = new ARMCounter32Metric(start);
ARMCntrDivr32Metric metric = new ARMCntrDivr32Metric(start, 1000);
```

**メトリック 7:**

**構文:**

```
myTransaction.setStringMetricData(text);
```

# UDM を使用しないトランザクションの設定

UDM を使用せずにトランザクションを設定する場合は、新しいトランザクションをすぐに作成できます。詳細を指定するかどうかを選択できます。

**詳細を使用する場合**

**構文:**

```
ARMTransaction myTransaction =
myApplication.createTransaction("Transactionname","details");
```

**詳細を使用しない場合**

**構文:**

```
ARMTransaction myTransaction =
myApplication.createTransaction("Transactionname");
```

# トランザクション インスタンスの設定

新しいトランザクション インスタンスを設定するには、ARMTransaction の `createTransactionInstance()` メソッドを使って、新しい ARMTransactionInstance のインスタンスを作成します。

**構文:**

```
ARMTransactionInstance myTranInstance =
myTransaction.createTransactionInstance();
```

# トランザクション インスタンスの開始 (arm_start)

トランザクション インスタンスを開始するには、相関係数を使用するかどうかを選択できます。次のメソッドは、関連するパラメータを使って C の関数 `arm_star` をコールします。

## 相関係数を使用したトランザクション インスタンスの起動

相関係数を使用する場合は、相関係数の取得と送信を区別する必要があります。

### 相関係数の要求

トランザクション インスタンスが相関係数を要求する場合は、コールは次のようになります (`jcorrelators.java` の例も参照)。

**構文:**

```
int status = myTranInstance.startTranWithCorrelator();
```

### 親相関係数の送信

以前のトランザクションの相関係数がすでに存在し、その相関係数をトランザクションに送信する場合は、構文は次のようになります。

**構文**

```
int status = startTran(parent);
```

**パラメータ**

`parent` は送信された相関係数です。以前のトランザクションでは、`getCorrelator()` メソッドを使ってトランザクション インスタンス相関係数を取得できます。

## 親相関係数の要求と送信

以前のトランザクションの相関係数がすでに存在し、その相関係数をトランザクションに送信して相関係数を要求する場合は、構文は次のようになります。

**構文:**

```
int status = myTranInstance.startTranWithCorrelator(parent);
```

**パラメータ:**

parent は送信された相関係数です。以前のトランザクションでは、getCorrelator() メソッドを使ってトランザクション インスタンス相関係数を取得できます。

## 相関係数情報の取得

getCorrelator() メソッドを使用して、トランザクション インスタンス相関係数を取得する場合は、次のようにします。

**構文:**

```
ARMTranCorrelator parent = myTranInstance.getCorrelator();
```

## 相関係数を使用しないトランザクション インスタンスの起動

相関係数を使用しない場合は、トランザクション インスタンスを次のように開始できます。

**構文:**

```
int status = myTranInstance.startTran();
```

startTran は、一意のハンドルをステータスに返します。これは、更新および停止に使用されます。

# トランザクション インスタンス データの更新

トランザクション インスタンスの UDM は、開始してから停止するまでの間に、何度でも更新できます。ラッパーのこの部分は、関連するパラメータを使って C の関数 arm_update をコールします。

## UDM を使用したトランザクション インスタンス データの更新

UDM を使用してトランザクション インスタンスのデータを更新する場合は、まずメトリックの新しいデータを設定する必要があります。たとえば、次のようになります。

ARM_Counter32 ARM_Counter64、ARM_Gauge32、ARM_Gauge64、ARM_NumericID32、ARM_NumericID64 の場合、metric.setData(value)

ARM_CntrDivr32、ARM_GaugeDivr32 の場合、metric.setData(value,value)

ARM_String8、ARM_String32 の場合、metric.setData(string)

これで、メトリック データを新規に設定でき (「メトリック データの設定」(361ページ)セクションの例を参照)、更新をコールできます。

**構文:**

```
myTranInstance.updateTranInstance();
```

## UDM を使用しないトランザクション インスタンス データの更新

UDM を使用しない場合は、更新をコールするだけでトランザクション インスタンスのデータを更新できます。これにより「ハートビート」が送信され、トランザクション インスタンスがまだ実行中であることが示されます。

**構文:**

```
myTranInstance.updateTranInstance();
```

# 大規模で非透過的なアプリケーション プライベート バッファの提供

2 番目のバッファフォーマットを使用する場合は、バイト アレイを更新メソッドに渡す必要があります(『*アプリケーション応答測定 2.0 API ガイド*』を参照)。

**構文:**

```
myTranInstance.updateTranInstance(byteArray);
```

# トランザクション インスタンスの停止 (arm_stop)

トランザクション インスタンスを停止するには、メトリック更新の有無を指定して停止をするかどうかを選択できます。

## メトリックを更新するトランザクション インスタンスの停止

メトリックを更新してトランザクション インスタンスを停止するには、`stopTranInstanceWithMetricUpdate` メソッドをコールします。

**構文:**

```
myTranInstance.stopTranInstanceWithMetricUpdate

transactionCompletionCode);
```

**パラメータ:**

トランザクション完了コードは、次のようになります。

| | | |
|---|---|---|
| `ARMConstants.ARM_GOOD.` | 操作が正常に実行された場合に使用します。 | `ARMConstants.ARM_GOOD.` |
| `ARMConstants.ARM_ABORT.` | システムで致命的なエラーが発生した場合に使用します。 | `ARMConstants.ARM_ABORT.` |
| `ARMConstants.ARM_FAILED.` | トランザクションは正常に機能したが、結果が生成されなかったアプリケーションで使用します。 | `ARMConstants.ARM_FAILED.` |

これらのメソッドは、C の関数 arm_stop に必要なパラメータを指定して使用します。

## メトリックを更新しないトランザクション インスタンスの停止

メトリックを更新せずにトランザクション インスタンスを停止するには、メソッド stopTranInstance を使用できます。

**構文:**

```
myTranInstance.stopTranInstance(transactionCompletionCode);
```

# 完全なトランザクションの使用

Java ラッパーは arm_complete_transaction コールを使用できます。このコールを使用すると、指定された時間 (ナノ秒) だけ継続したトランザクションの終了をマーキングできます。これにより、ARM エージェント以外で測定されたトランザクション応答時間をリアルタイムに統合できます。

トランザクション インスタンスの終了を通知するだけでなく、トランザクション (UDM) に関する追加情報もオプションのデータ バッファで提供されます

(jcomplete.java の例も参照してください)。

## UDM を使用した完全なトランザクションの使用

**構文:**

```
myTranInstance.completeTranWithUserData(status,responseTime;
```

**パラメータ:**

| | |
|---|---|
| status | • ARMConstants.ARM_GOOD<br>操作が正常に実行された場合に使用します。<br>• ARMConstants.ARM_ABORT<br>システムで致命的なエラーが発生した場合にこの値を使用します。<br>• ARMConstants.ARM_FAILED<br>トランザクションは正常に機能したが、結果が生成されなかったアプリケーションで使用します。 |
| responseTime | トランザクションの応答時間がナノ秒単位で表示されます。 |

## UDM 使用しない完全なトランザクションの使用

**構文:**

```
myTranInstance.completeTran(status,responseTime);
```

# その他のドキュメント

Java クラスについての詳細は、/<InstallDir>/examples/arm/ ディレクトリの doc フォルダを参照してください。ここには、すべての Java クラスが html 文書で含まれています。index.htm から始めます。

# 記録とトレース

記録とトレースの機構を使用すると、HP Operations エージェントの問題を診断してトラブルシューティングを実行できます。簡単に分析できるように、HP Operations エージェントではエラー、警告、一般メッセージをログファイルに保存します。

トレース機構は、エージェントのオペレーションで発生した特定の問題をトレースするときに役立ちます。また、トレース機構で生成されたトレースファイルを HP サポートに送付して、さらに分析することもできます。

## 記録

HP Operations エージェントでは、警告メッセージ、エラーメッセージ、情報通知をノードの `System.txt` ファイルに書き込みます。`System.txt` ファイルの内容により、エージェントが正常に機能しているかどうかを判断できます。`System.txt` ファイルは次の場所にあります。

*Windows の場合*

*%ovdatadir%*`log`

*UNIX/Linux の場合*

`/var/opt/OV/log`

さらに、HP Operations エージェントは Performance Collection Component と coda のステータスの詳細を次のファイルに追加します。

*Windows の場合*

- *%ovdatadir%*`\status.scope`
- *%ovdatadir%*`\status.perfalarm`
- *%ovdatadir%*`\status.ttd`
- *%ovdatadir%*`\status.mi`
- *%ovdatadir%*`\status.perfd-`*<ポート>*

**ヒント:** この場合、*<ポート>* は `perfd` で指定したファイル名です。デフォルトでは、`perfd` はポート 5227 を使用します。`perfd` のデフォルト ポートを変更するには、Configuring the RTMA Component on page 48を参照してください。

*%ovdatadir%*`\log\coda.txt`

*UNIX/Linux の場合*

- `/var/opt/perf/status.scope`
- `/var/opt/perf/status.perfalarm`
- `/var/opt/perf/status.ttd`
- `/var/opt/perf/status.mi`
- `/var/opt/perf/status.perfd`

- *vMA のみ。* `/var/opt/perf/status.viserver`
- `/var/opt/OV/log/coda.txt`

## 記 録ポリシーの設 定

`System.txt` のファイル サイズが 1 MB になると、エージェントは新しい `System.txt` ファイルにメッセージを記 録し始 めます。`System.txt` ファイルのサイズを制 限 するように HP Operations エージェントのメッセージ記 録ポリシーを設 定 できます。

デフォルト の記 録ポリシーを修 正 するには

デフォルト の記 録ポリシーを変 更 するには、次 の手 順を実 行します。

1. ノード にログオンします。
2. 以 下 の場 所 に移 動します。
   *Windows の場 合*

   *%ovdatadir%*`conf\xpl\log`

   *UNIX/Linux の場 合*

   `/var/opt/OV/conf/xpl/log`
3. テキスト エディタで `log.cfg` ファイルを開きます。
4. `System.txt` ファイルのバイト サイズと文 字 数を制 御しているパラメータは、`BinSizeLimit` および `TextSizeLimit` です。デフォルト では、両 方 のパラメータは 1000000 (1 MB および 1000000 文 字 ) に設 定されています。目 的 の値 にデフォルト 値を変 更します。
5. ファイルを保 存します。
6. 次 のコマンド で、運 用 監 視コンポーネントを再 起 動します。
   a. `ovc -kill`

   b. `ovc -start`

## トレース

HP Operations エージェント アプリケーションのトレースを開 始 する前 に、事 前 に必 要 な作 業を実 行 する必 要 があります。これには、トレースする適 切 なアプリケーションの特 定 、トレースのタイプの設 定 、( 必 要 に応 じて)トレース設 定 ファイルの生 成 などがあります。

HP Operations エージェント プロセスのトレースを開 始 する前 に、次 の作 業を実 行します。

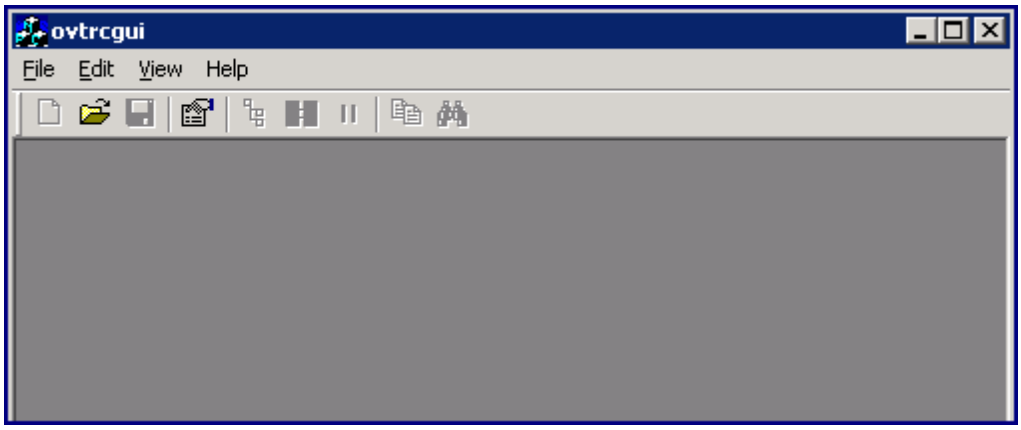
1. アプリケーションの特 定
2. トレース タイプの設 定
3. *オプション:* 設 定 ファイルの作 成

## アプリケーションの特 定

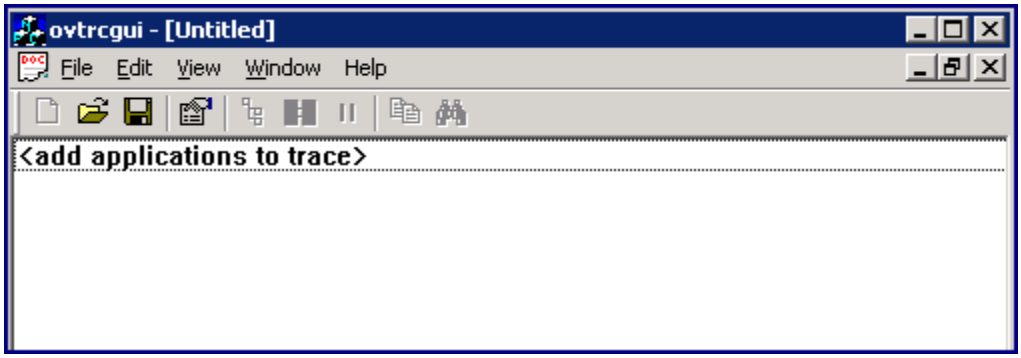
トレースする HP ソフトウェア アプリケーションを、管 理 対 象 システムで特 定します。ovtrccfg -vc オプションを使 用して、すべてのトレース対 応 アプリケーションの名 前 、および各トレース対 応 アプリケーションに定 義 されているコンポーネントとカテゴリを表 示します。

または、ovtrcgui ユーティリティを使用して、トレース対応アプリケーションのリストを表示できます。ovtrcgui ユーティリティを使用して、トレース対応アプリケーションのリストを表示するには、次の手順を実行します。

1. `ovtrcgui.exe` ファイルを *%OvInstallDir%*`\support` ディレクトリから実行します。ovtrcgui ウィンドウが表示されます。

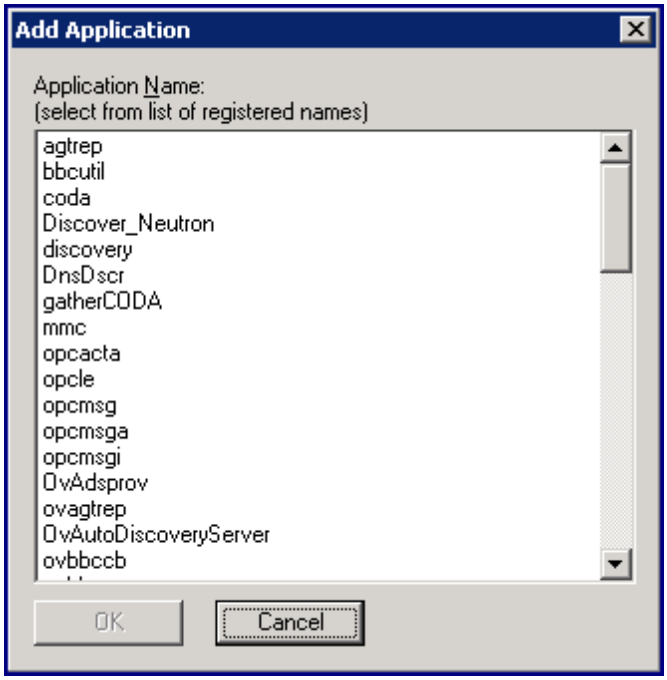



2. ovtrcgui ウィンドウで **[ファイル]** → **[新規]** → **[トレースの設定]** をクリックします。新しいトレース設定エディタが開きます。




3. ovtrcgui ウィンドウで **[編集]** → **[Add Application (アプリケーションを追加)]** をクリックします。または、エディタを右クリックし、**[Add Application (アプリケーションを追加)]** をクリックします。[Add

Application (アプリケーションを追 加 )] ウィンド ウが表 示 されます。




[Add Application (アプリケーションを追 加 )] ウィンド ウに、利 用 できるトレース対 応 アプリケーションのリストが表 示 されます。

## トレース タイプの設 定

トレース機 構を有 効 にする前 に、アプリケーションに設 定 するトレースのタイプを決 定 して設 定 します。トレースのタイプを設 定 するには、次 の手 順 を実 行 します。

設 定 するトレースのタイプ (静 的 または動 的 ) を決 定 し、次 の手 順 を実 行 します。

1. <*データ ディレクトリ*>`/conf/xpl/trc/` の場 所 に移 動 します。
2. <*アプリケーション名* >`.ini` ファイルを見 つけます。ファイルがあれば、手 順 3 に進 みます。<*アプリケーション名* >`.ini` ファイルがなければ、次 の手 順 を実 行 します。
   - テキスト エディタで新 しいファイルを作 成 します。
   - ファイルに `DoTrace`、`UpdateTemplate`、`DynamicTracing` の各 プロパティを、この順 番 で追 加 します。

     **ヒント :** すべてのプロパティを 1 行 に記 述 せず、各 プロパティを新 しい行 に記 述 してください。たとえば、次 のようになります。

     ```
     DoTrace=
     UpDateTemplate=
     DynamicTracing=
     ```

   - ファイルを保 存 します。
3. <*アプリケーション名* >`.ini` ファイルをテキスト エディタで開 きます。

4. 静的なトレースを有効にするには、DoTrace プロパティを ON に設定し、DynamicTracing プロパティを OFF に設定します。
5. 動的なトレースを有効にするには、DoTrace プロパティと DynamicTracing プロパティを ON に設定します。
6. UpdateTemplate プロパティを ON に設定します。
7. ファイルを保存します。

動的トレース設定の場合、アプリケーションを起動した後でもトレース機構を有効にできます。静的トレース設定の場合、アプリケーションを起動する前にトレース機構を有効にする必要があります。

# トレース設定ファイルの概要

## 構文

TCF バージョン<*バージョン番号*>

APP: "<*アプリケーション名*>"

SINK: File "<*ファイル名*>" "maxfiles=[*1..100*];maxsize=[*0..1000*];"

TRACE: "<*コンポーネント名*>" "<*カテゴリ名*>" <*キーワードのリスト*>

構文の各行については、この後の章で説明します。

## 設定ファイルの作成

設定ファイルを使用せずにトレース機構を有効にする場合、この項を省略して後述の「コマンド ライン ツールを使用したトレースの有効化とトレース メッセージの表示」に進みます。

トレース設定ファイルは、コマンド ライン ツールの ovtrccfg、テキスト エディタ、ovtrcgui ユーティリティ (Windows ノードのみ) のいずれかを使用して作成できます。

# コマンド ライン ツールを使用したトレースの有効化とトレース メッセージの表示

次に説明する手順では、トレースを有効にするために必要な一般的な手順を取り上げます。トレース機構を有効にするには、次の手順を実行します。

1. ovtrccfg を使用して、トレース設定要求を行います。
   ovtrccfg -cf <*設定ファイル名*>

   ここで、<*設定ファイル名*> は、「*設定ファイルの作成*」で作成されたトレース設定ファイルの名前です。

   > **注 :** トレース設定ファイルを使用しない場合、次のコマンドでトレースを有効にできます。
   >
   > ovtrccfg -app <*アプリケーション*>[-cm <*コンポーネント*>]

2. 静的トレース機構を設定する場合、トレースするアプリケーションを起動します。

3. トレースする問題を再現するために必要な、アプリケーション固有のコマンドを実行します。目的の動作が再現されたら、トレースを停止できます。
4. `ovtrcmon` を使用して、トレース監視要求を行います。
   トレースメッセージを監視するには、次のコマンドの 1 つを実行するか、または追加の `ovtrcmon` コマンド オプションを使用して同様のコマンドを実行します。
   - `/opt/OV/bin/trace1.trc` からトレースメッセージを監視し、トレースメッセージをテキスト形式でファイルに送信するには、次を実行します。
     ```
     ovtrcmon -fromfile /opt/OV/bin/trace1.trc -tofile
     /tmp/traceout.txt
     ```
   - `/opt/OV/bin/trace1.trc` から、詳細形式でトレースメッセージを表示するには、次を実行します。
     ```
     ovtrcmon -fromfile /opt/OV/bin/trace1.trc -verbose
     ```
   - `/opt/OV/bin/trace1.trc` から、詳細形式でトレースメッセージを表示し、トレースメッセージをファイルに出力するには、次を実行します。
     ```
     ovtrcmon -fromfile /opt/OV/bin/trace1.trc -short >
     /tmp/traces.trc
     ```
5. `ovtrccfg` を使用してトレースを停止または無効にするには、次のコマンドを実行します。
   ```
   ovtrccfg -off
   ```
6. トレース設定ファイルとトレース出力ファイルを収集します。トレースメッセージを評価するか、またはファイルをパッケージ化して、評価のために HP ソフトウェア サポート オンラインに送付します。システムにはトレース出力ファイルの複数のバージョンがあります。`Maxfiles` オプションを使用すると、トレース機構で複数のトレース出力ファイルを生成できます。これらのファイルには、`.trc` の拡張子と、サフィックス `n` が付いています (ここで `n` は `1` から`99999` の整数です)。

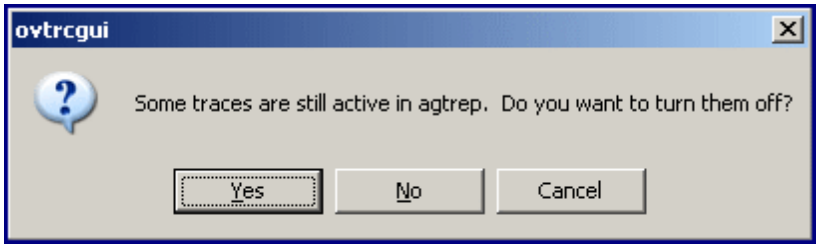
# トレース GUI を使用したトレースの有効化とトレース メッセージの表示

Windows ノードでは、`ovtrcgui` ユーティリティを使用して、トレースを設定し、トレースメッセージを表示できます。

## トレース機構の有効化

トレース設定ファイルを使用せずに、`ovtrcgui` ユーティリティを使用してトレース機構を有効にするには、次の手順を実行します。

1. Using the Tracing GUIの Step 1 から Step 6 を実行します。
2. トレース設定エディタを閉じます。
3. 無題への変更を保存するかどうかを尋ねるプロンプトが表示されたら、**[いいえ]** をクリックします。表示されるメッセージに従います。



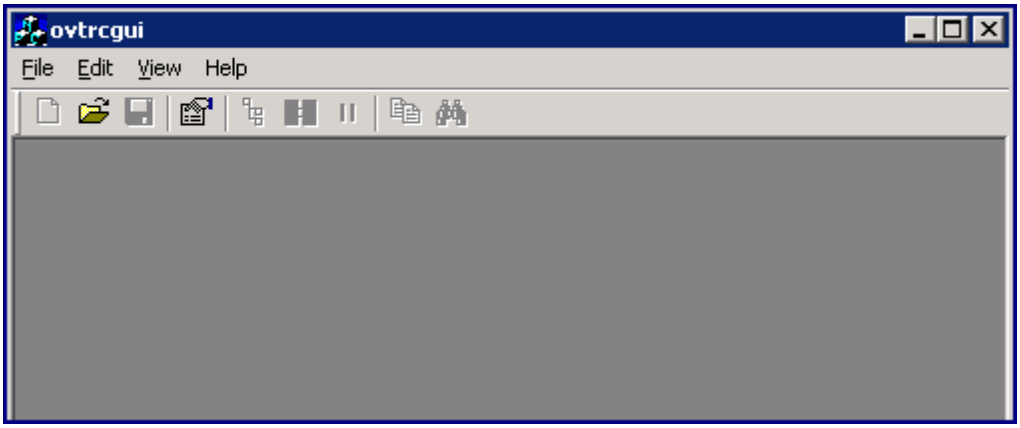

4. **[いいえ]** をクリックします。**[はい]** をクリックすると、`ovtrcgui` ユーティリティは直ちにトレース機構を無効にします。

`ovtrcgui` ユーティリティとトレース設定ファイルを使用してトレース機構を有効にするには、トレース設定ファイルがあるローカルシステムの場所に移動し、トレース設定ファイルをダブルクリックします。または、`ovtrcgui` ユーティリティを開き、**[ファイル]** → **[開く]** をクリックして、トレース設定ファイルを選択し、**[開く]** をクリックします。

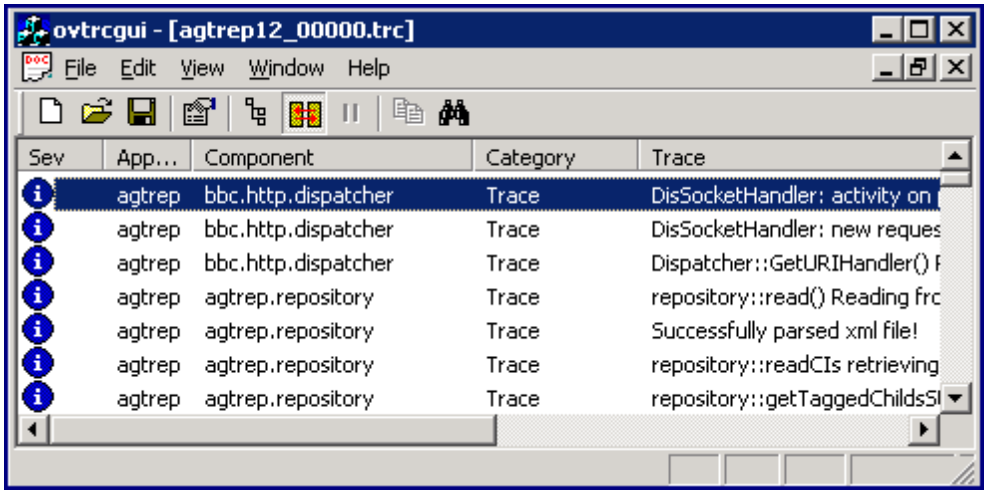
## トレース メッセージの表示

`ovtrcgui` ユーティリティでトレース出力ファイルを表示するには、次の手順を実行します。

1. `ovtrcgui.exe` ファイルを *%OvInstallDir%*`\support` ディレクトリから実行します。ovtrcgui ウィンドウが表示されます。

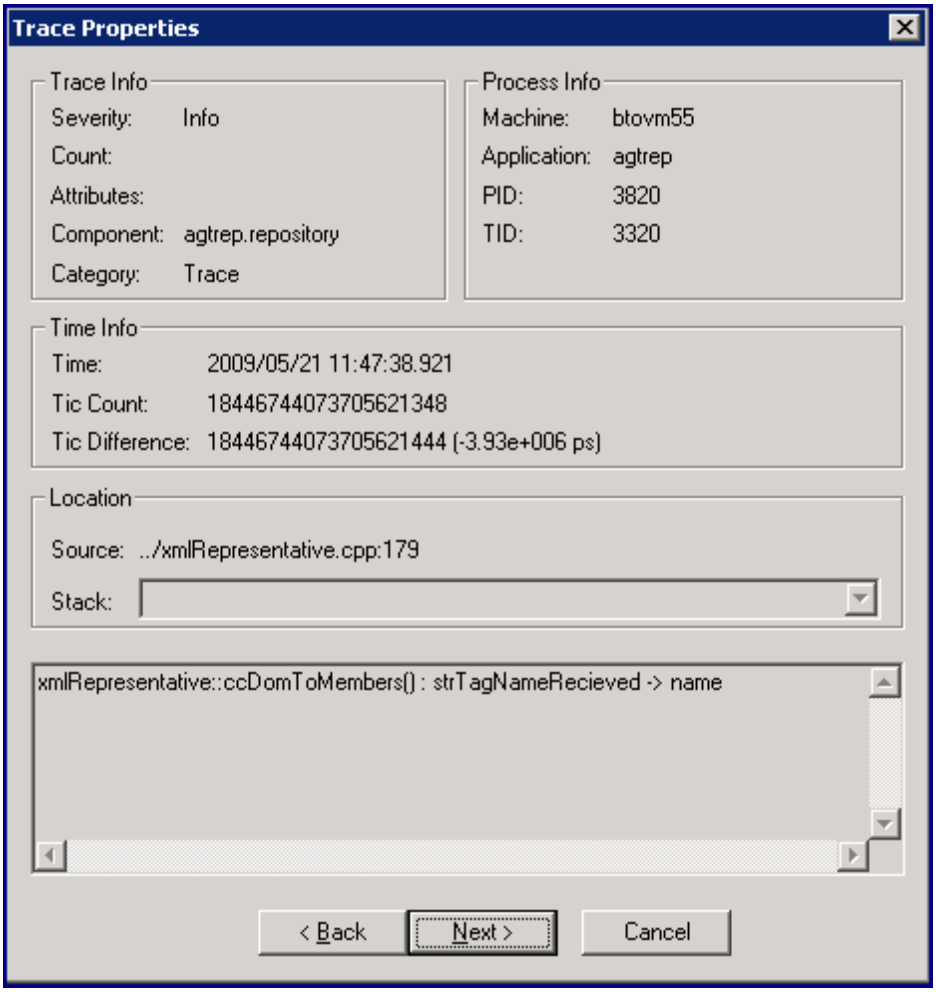



2. **[ファイル]** → **[開く]** をクリックします。[開く] ダイアログ ボックスが表示されます。
3. トレース出力ファイルが配置されている場所に移動し、`.trc` ファイルを選択して、**[開く]** をクリックします。ovtrcgui ユーティリティに `.trc` ファイルの内容が表示されます。




`.trc` ファイルの新しい行 は、それぞれ新しいトレース メッセージを表します。

4. トレース メッセージをダブルクリックして、詳 細を表 示します。[Trace Properties (トレース プロパティ)] ウィンド ウが表 示されます。




[Trace Properties (トレース プロパティ)] ウィンド ウに、次 の詳 細 が表 示されます。

- [Trace Info (トレース情報)]:
  - *[重要度]:* トレースメッセージの重要度です。
  - *[Count (カウント)]:* メッセージのシリアル番号です。
  - *[属性]:* トレースメッセージの属性です。
  - *[コンポーネント]:* トレースメッセージを発行したコンポーネントの名前です。
  - *[カテゴリ]:* トレース対象アプリケーションによって割り当てられた任意の名前です。
- [Process Info (プロセス情報)]:
  - *[Machine (マシン)]:* ノードのホスト名です。
  - *[アプリケーション]:* トレース対象アプリケーションの名前です。
  - *[PID]:* トレース対象アプリケーションのプロセス ID です。
  - *[TID]:* トレース対象アプリケーションのスレッド ID です。
- [Time Info (時間情報)]:
  - *期間:* トレースメッセージの現地での日時です。
  - *[Tic count (Tic カウント)]:* 高分解能な経過時間です。
  - *[Tic difference (Tic 差)]:*
- [Location (場所)]
  - *[ソース]:* トレースを生成するソースの行番号とファイル名です。
  - *[Stack (スタック)]:* トレース対象アプリケーションの呼び出しスタックの説明です。

5. **[次へ]** をクリックして、次のトレースメッセージを表示します。
6. すべてのトレースメッセージを表示したら、**[キャンセル]** をクリックします。

## トレース リスト ビューの使用

デフォルトでは、`ovtrcgui` ユーティリティはトレース リスト ビューにあるトレース ファイルのトレースメッセージを表示します。トレース リスト ビューは、トレースメッセージを表形式で表示します。

このトレース リスト ビューには、すべてのトレースメッセージが次の列と共に表示されます。

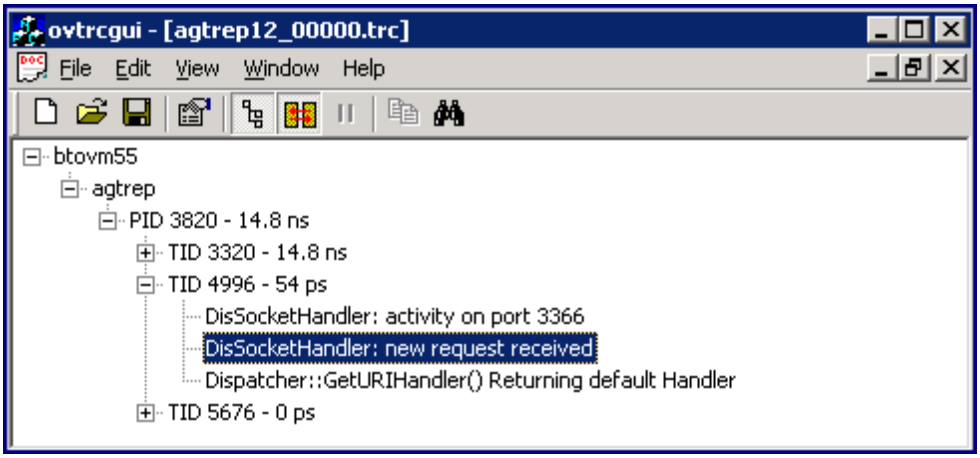
**表 13:トレース リスト ビュー**

| カラム | 説明 | |
|---|---|---|
| 重要度 | トレースメッセージの重要度を示します。このビューでは次のアイコンを使用して、メッセージの重要度を表示します。 | |
| | 情報 | |
| | 警告 | |
| | エラー | |

| | |
|---|---|
| アプリケーション | トレース対象アプリケーションの名前を表示します。 |
| コンポーネント | トレースメッセージを生成したトレース対象アプリケーションのコンポーネントの名前を表示します。 |
| カテゴリ | トレースメッセージのカテゴリを表示します。 |
| トレース | トレースメッセージテキストを表示します。 |

## 手順のツリービューの使用

手順のツリービューでは、構造化された形式でトレースメッセージを表示できます。手順のツリービューでは、プロセス ID とスレッド ID に基づいてメッセージが並び替えられ、ツリービューの形式でデータが表示されます。

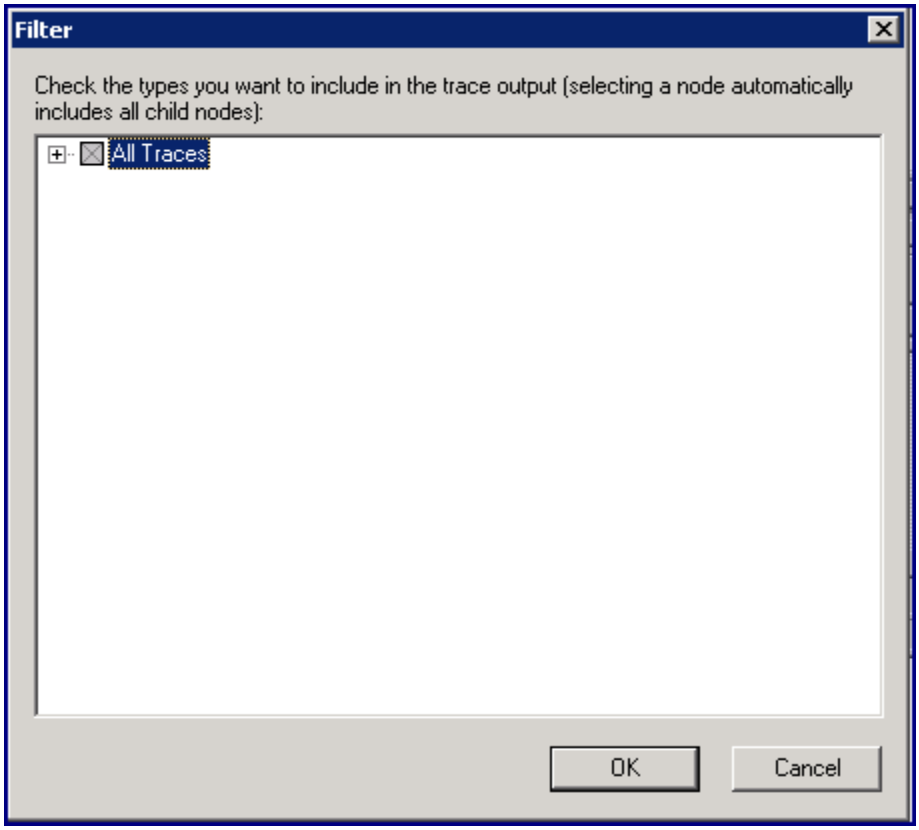



プロセス ID とスレッド IDを展開して、トレースメッセージを表示できます。トレースリストビューに戻るには、 をクリックします。

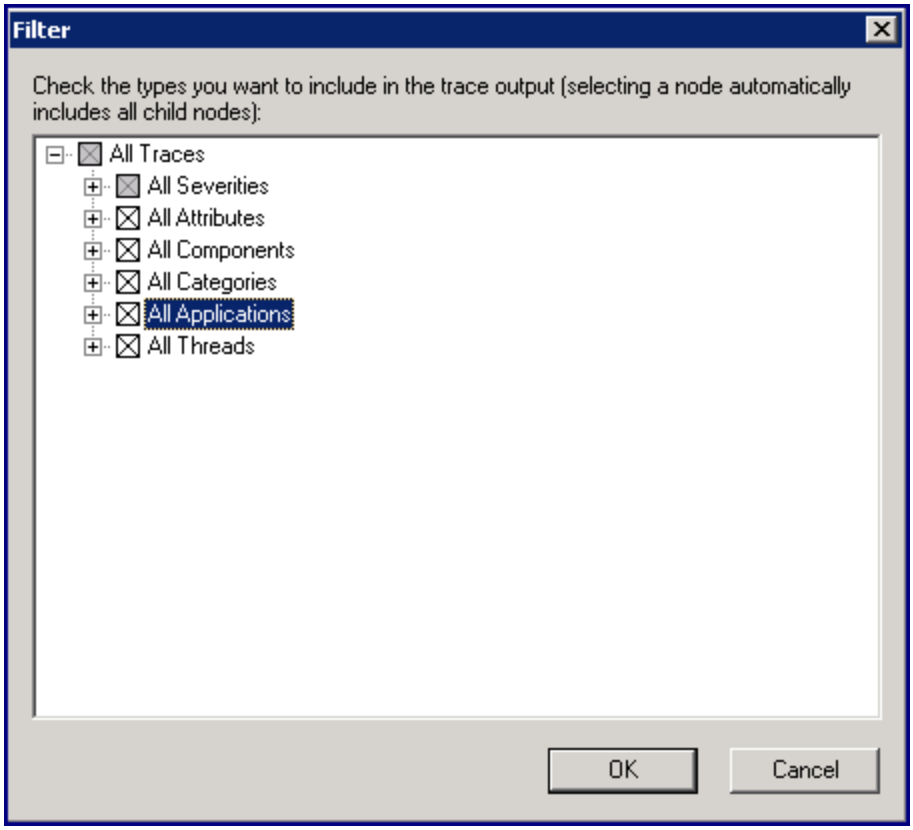
## トレースのフィルタリング

ovtrcgui ユーティリティは、トレース設定ファイルの設定に基づいてトレース出力ファイルに記録されたすべてのトレースメッセージを表示します。利用可能なメッセージをフィルタリングすると、ovtrcgui コンソールに目的のメッセージのみを表示できます。利用可能なトレースメッセージをフィルタリングするには、次の手順を実行します。

1. ovtrcgui コンソールで **[表示]** → **[フィルタ]** をクリックします。[フィルタ] ダイアログボックスが表示されます。




2. **[すべてのトレース]** を展 開します。ダイアログ ボックスに、すべてのフィルタリング パラメータがツリー形 式 で表 示 されます。




3. パラメータを展 開して、トレース メッセージをフィルタリングする項 目 を選 択します。
4. **[OK]** をクリックします。ovtrcgui コンソールにはフィルタリングされたメッセージのみが表 示 されます。

# トラブルシューティング操作

ここでは、HP Operations エージェントで作業するときによく見られる問題の対策または回避策について説明します。ここで扱われている範囲は以下のとおりです。

- 運用監視コンポーネント
- Performance Collection Component
- RTMA

# 運用監視コンポーネント

- *問題:*HPOM が多数 (1024 超) のノードを管理している場合、HPOM と管理対象ノード間で通信の問題が発生する場合がある。また、多数 (1024 超) の管理対象ノードと通信を行う HP Performance Manager に HP Operations エージェントがインストールされている場合も、この問題が発生することがある。

  *対策:*

  この問題を回避するには、管理サーバー (HPOM が 1024 を超えるノードを管理している場合) または HP Performance Manager サーバー (1024 を超えるノードと通信を行う HP Performance Manager サーバーにエージェントがインストールされている場合) に移動し、次の構成手順を実行します。

  root または管理者としてログオンします。

  次のコマンドを実行します。

  *Windows の場合*

  ```
  %ovinstalldir%bin\ovconfchg -ns xpl.net -set SocketPoll true
  ```

  *UNIX/Linux の場合*

  ```
  /opt/OV/bin/ovconfchg -ns xpl.net -set SocketPoll true
  ```

  エージェントを再起動します。

  *Windows の場合*

  ```
  %ovinstalldir%bin\opcagt -stop
  %ovinstalldir%bin\opcagt -start
  ```

  *HP-UX、Linux、Solaris の場合*

  ```
  /opt/OV/bin/opcagt -stop
  /opt/OV/bin/opcagt -start
  ```

- *問題:*Windows Server 2008 ノードで、`opcmsga` プロセスが機能せず、`ovc` コマンドを使用すると `opcmsga` プロセスのステータスが `aborted` と表示される。
  *対策:*

  以下のコマンドを実行して `OPC_RPC_ONLY` 変数を `TRUE` に設定します。

  ```
  ovconfchg -ns eaagt -set OPC_RPC_ONLY TRUE
  ```

- *問題:*Windows ノードで、Perl スクリプトがポリシーから機能しない。
  *原因:*ポリシーで Perl スクリプトを利用するには、`PATH` 設定変数に (HP Operations エージェントで提供される) Perl があるディレクトリを設定する必要がある。

  *対策:*

  a. 以下のコマンドを実行して、`PATH` 設定変数を Perl ディレクトリに設定します。
     `ovconfchg -ns ctrl.env -set PATH "%ovinstalldir%nonOV\perl\a\bin"`

  b. 以下のコマンドを実行してエージェントを再起動します。
     i. `ovc -kill`

     ii. `ovc -start`

- *問題:*`ovconfchg` コマンドから変数の値を変更した後、その変更が有効にならない。
  *原因 1:*

  その変数はエージェントを再起動する必要がある。

  *対策 1:*

  以下のコマンドを実行してエージェントを再起動します。

  a. `ovc -kill`

  b. `ovc -start`

  *原因 2:*

  ノードに配布されている ConfigFile ポリシーにより、その変数が特定の値に設定される。

  *対策 2:*

  配布されている ConfigFile ポリシーに、設定変数を特定の値に設定するコマンドが含まれている場合、`ovconfchg` コマンドからの変更は有効になりません。ConfigFile ポリシーをノードから削除するか、または変数を目的の値に設定するコマンドを含めるようにポリシーを変更する必要があります。

  *原因 3:*

  ノードで利用可能なプロファイルまたはジョブファイルによって、変更が上書きされる。

  *対策 3:*プロファイルまたはジョブファイルをノードで開き、それらのファイルに競合する変数の設定が含まれていないことを確認します。

- *問題:*設定変数 `SNMP_SESSION_MODE` の値を変更した後、`ovc` によって `opctrapi` プロセスのステータスが `Aborted` と表示される。
  *原因:*

  設定変数 `SNMP_SESSION_MODE` の値を変更した後、HP Operations エージェントは `opctrapi` を再起動しようとする。場合によっては、`opctrapi` の再起動プロセスが失敗することがある。

  *対策:*

  以下のコマンドを実行して `opctrapi` を再起動します。

  `ovc -start opctrapi`

- *問題:*ノード上で組み込み Perl スクリプトを使用してスケジュールタスクポリシーを実行すると、`opcmona` プロセスが自動的に再起動され、HPOM コンソールに次のメッセージが表示され

る。

```
(ctrl-208) Component 'opcmona' with pid 6976 exited with exit value
'-1073741819'.Restarting component.
```

*原因:*

組み込み Perl スクリプトの `exit (0)` の参照により `opcmona` が再起動する。

*対策:*

組み込み Perl スクリプトに `exit (0)` を使用しないでください。

# Performance Collection Component

- *問題:*HP-UX 11.11 システムで `status.midaemon` ファイルに次のエラーが表示される。
  `mi_shared - MI initialization failed (status 28)`

  *原因:*`midaemon` バイナリのページ サイズが大きい。

  *対策:*この問題を解決するには、以下の手順に従います。

  a. root ユーザーとしてシステムにログオンします。

  b. 以下のコマンドを実行して、HP Operations エージェントを停止します。

  ```
  /opt/OV/bin/opcagt -stop
  ```

  c. 以下のコマンドを実行して、`midaemon` のバックアップを作成します。
  `cp /opt/perf/bin/midaemon /opt/perf/bin/midaemon.backup`

  d. 以下のコマンドを実行して、`midaemon` バイナリのページ サイズを 4K に縮小します。
  `chatr +pi 4K /opt/perf/bin/midaemon`

  e. 以下のコマンドを実行して、HP Operations エージェント を起動します。
  `/opt/OV/bin/opcagt -start`

- *問題:*HP Operations エージェントをインストールした後、トレース機構が有効になっていると `System.txt` ファイルに以下のエラー メッセージが表示される。
  `Scope data source initialization failed`

  *対策:*このエラーは無視してください。

- *問題:*以下のエラー メッセージが HPOM コンソールに出力される。

  ```
  CODA:GetDataMatrix returned 76='Method
  ScopeDataView::CreateViewEntity failed
  ```

  *原因:*ソースが組み込みパフォーマンス コンポーネントに設定されている Measurement Threshold ポリシーで、`PROCESS` オブジェクトを `SCOPE` データ ソースとともに使用している場合に、このエラーが出力される。

  *対策:*代わりに、サービス/プロセス監視ポリシーを使用します。

- *問題:HP Performance Manager や HP Service Health Reporter といったデータ分析用製品がエージェントのデータ ストアからのデータの取得に失敗し、以下のエラーが表示される。*

  ```
  Error occurred while retrieving data from the data source
  ```

*原因:* エージェントのデータ アクセス ユーティリティは、クライアント プログラムからの単一のクエリに対して、データ クラスのすべてのレコードを読み込む。クエリは、HP Performance Manager といったデータ分析クライアントによってエージェントに送信される。データ クラスに大量のレコードが含まれていると、データ アクセス ユーティリティがクエリの処理に失敗する。

*対策:* この問題を回避するには、データ レコードを複数のチャンクで転送するようデータ アクセス ユーティリティを設定します。以下の手順に従ってください。

a. root または管理者として、エージェント ノードにログオンします。

b. 次のコマンドを実行します。
*Windows の場合:*

*%ovinstalldir%*`bin\ovconfchg -ns coda -set DATAMATRIX_VERSION 1`

*HP-UX、Solaris、Linux:*

```
/opt/OV/bin/ovconfchg -ns coda -set DATAMATRIX_VERSION 1
```

*AIX の場合:*

```
/usr/lpp/OV/bin/ovconfchg -ns coda -set DATAMATRIX_VERSION 1
```

各クエリに対して、エージェントのデータ アクセス ユーティリティがデータを 5 つのレコードのチャンクに分割し、クライアント プログラムにデータを送信します。データをチャンクに分割すると、データ転送プロセスのパフォーマンスが向上します。

エージェントが1つのチャンクでクライアントに送信できるレコードの数を制御できます。`DATAMATRIX_ROWCOUNT` 変数 (`coda` 名前空間で使用可能な変数) により、この数を制御できます (デフォルト値は 5)。

`DATAMATRIX_ROWCOUNT` 変数の値を減らすと、データ ストアに大量のデータがある場合に、データ転送率がわずかに向上することがあります。

`DATAMATRIX_ROWCOUNT` 変数を 0 に設定した場合は、HP Operations エージェントはチャンクなしでデータ レコードを送信するデフォルトの動作に戻ります。

ただし、`DATAMATRIX_ROWCOUNT` 変数のデフォルト設定を変更しないことをお勧めします。

c. エージェントを再起動して設定変更を有効にします。
`ovc -restart coda`

# RTMA

- *問題:*vSphere Management Assistant (vMA) ノードで、`rtmd` プロセスが機能せず、`ovc` コマンドを使用すると `rtmd` プロセスのステータスが `aborted` と表示される。
*原因:*`rtmd` プロセスが IP アドレスに対してシステムのホスト名を解決できない。

*対策:*

a. root 権限を使用して、ノードにログオンします。

b. テキスト エディタで、`/etc` ディレクトリから `hosts` ファイルを開きます。

c. `localhost` という文字が記されている行を見つけます。

d. 行頭の `#` 文字を削除します。

e. ファイルを保存します。

f. 以下のコマンドを実行し、すべてのプロセスを起動します。

```
ovc -restart
```

- *問題:*HP Performance Manager の診断ビューでデータにアクセスできない。
  *原因:*`rtmd` プロセスが実行されていない。

  *対策:*`rtmd` プロセスが HP Operations エージェントノードで実行されているかどうかを確認するには、`ovc -status rtmd` を実行します。`rtmd` プロセスを起動するには、`ovc -start rtmd` を実行します。

- *問題:*HP-UX 11.11 システムで `status.perfd` ファイルに次のエラーが出力される。
  `mi_shared - MI initialization failed (status 28)`

  *原因:*`perfd` バイナリのページ サイズが大きい。

  *対策:*この問題を解決するには、以下の手順に従います。

  a. root ユーザーとしてシステムにログオンします。

  b. 以下のコマンドを実行して、HP Operations エージェントを停止します。
  `/opt/OV/bin/opcagt -stop`

  c. 以下のコマンドを実行して、`perfd` のバックアップを作成します。
  `cp /opt/perf/bin/perfd /opt/perf/bin/perfd.backup`

  d. 以下のコマンドを実行して、`perfd` バイナリのページ サイズを 4K に縮小します。
  `chatr +pi 4K /opt/perf/bin/perfd`

  e. 以下のコマンドを実行して、HP Operations エージェントを起動します。
  `/opt/OV/bin/opcagt -start`